

# Computer Science NEA AQA



**[expense OCR tracker]**

Zubaid Amadxarif  
2020

## Table of Contents

Analysis.....	4
Introduction .....	4
Project Background .....	4
Project Brief.....	4
Research.....	5
Client Research .....	5
Analysis of Interview 1 .....	6
Current Systems.....	7
Shoeboxed and Neat .....	7
Optical Character Recognition.....	10
Regular expressions .....	12
Analysis of Interview 2 .....	13
Proposed System.....	13
Input Data.....	14
OCR extraction – Input Data .....	14
Manual Receipt Input .....	15
Manual Sales Input .....	15
Edit Data .....	16
Generate Reports .....	16
SQL Statements .....	17
SQL Statements .....	18
Different Receipt Types .....	19
Adaptive Binarization with High Threshold .....	20
Loss Sheet.....	20
End User System.....	21
Smart Objectives .....	21
Modelling the solution .....	24
Data Flow Diagrams .....	25
Level 0 .....	25
Level 1 .....	26
System Flowchart.....	27
Final Flowchart.....	27
Log In .....	28
Edit Data .....	28
Input Receipt/Sales .....	29
Generate Reports .....	30

ERD .....	31
Project Approach .....	32
Documented Design .....	34
Overall System Design .....	34
Top Down Chart.....	34
Data Dictionary .....	34
Data Structures .....	35
Queues .....	35
User Interface Design .....	36
Log-In System.....	36
Main Program .....	38
Input Receipts and Sales.....	38
Input Sales .....	39
Costs.....	41
Edit Data .....	43
Generate Reports .....	45
Data handling.....	47
Entity Attribute Model .....	47
First Normal Form .....	48
Second Normal Form .....	48
Third Normal Form .....	48
Post-Normalisation Entity Relationship Diagram .....	49
Data Dictionaries for tables .....	50
SQL Statements .....	53
Technical Solution .....	54
Folder Management .....	54
Database Management.....	56
Initial Program Run – Log in Page – MainProgram.py.....	57
ChoiceOfThree.py .....	59
InputData.py .....	60
Edit Data.py .....	72
GenerateReport.py.....	78
Testing.....	90
Changes made to code .....	94
Test 4.4 – Input Data .....	94
Error 2 – PlaceIndex.....	94
Error 4 – Integer String .....	95
Test 6, 6.2, 6.2 – Input Sales .....	95

Evaluation .....	96
Objective Comparison .....	96
Client Feedback .....	98
References .....	101
Harvard Reference Table.....	101
Appendices .....	<b>Error! Bookmark not defined.</b>
Appendix 1 – Client Interview 1 .....	<b>Error! Bookmark not defined.</b>
Appendix 2 – Client Interview 2.....	<b>Error! Bookmark not defined.</b>

## Analysis

### Introduction

#### Project Background

Receipts have been a way of tracking expenditures for a lot of years. Many sole traders do this to keep proof of their sales so that they can hand over their paperwork to the legal authorities. Over the years, accountants have been taking note of this for sole traders and providing them with the legal documents that are needed.

Keeping large mass of receipts has become a problem for many sole traders. Some of whom do not understand how to keep track of their data. Their data is manually sorted, written in a table and then handed over to the necessary. Manual paperwork in the age of technology is not the best option for such sole traders. Hence, such sole traders may need software which can keep track of the money that they made. Technology has aged into AI affecting all aspects of our lives. Technologies such as Image Classification and Character Recognition can be used to scrape data automatically which could cut down all the time taken for the project.

#### Project Brief

I will be creating a tool which will allow my client to keep track of his expenditures by allowing an AI to read his receipts and create a balance sheet / money tracker. Instead of his accountant going through and adding up the costs, one by one, my tool will be able to keep track with all costs and sales that have been made. This also eliminates human error as an AI will be doing the menial task.

This tool will also have an area for Mr. Saleh to input each sale he makes which can then calculate his profits made then and there. This will hopefully cut down a 2-month process to a day. This will save a lot of time and it will also keep my client aware of where his money goes and how much he makes, before his income support arrives.

Mr. Saleh will also be able to cut down costs as he will not need his accountant to do the task for him. Sole traders do not **need** accountants, which means that he has more profits to reinvest into the business or use for his own leisure. With the rise in inflation, and my client keeping his prices fairly the same he will have to use such methods to make his life easier alongside higher profits.

## Research

### Client Research

Mr. Yunus Saleh is a sole trader who makes clothes and sells it to his family and friends. He does this for a living and has done so for the past 20 years, alongside having the same accountant.

Being a sole trader means that he is seen as the same as his business so anything he buys, must be noted as a business transaction. He keeps all his receipts and hand these over to his accountant who manually makes note of all my client's profits who then hands this over to the government. Due to low sales, my client gets income support based on how much he makes which affects his everyday life.

When his documents (sales and costs) are handed over to his accountant, his accountant calculates his overall profits which is then handed over to the government as a business record. This process could take up to 2 months - which is a very long time for Mr. Saleh to wait.

We discussed his problem, file management and what his perfect solution would be when I went to visit him at his place of work. Below is the extract of the first interview.

### Interview 1

#### ***-interview begins-***

ZA: Let's jump right into the questions. What do you work as?

YS: I am a sole trader, and I work as a tailor. I have done so for about 20 years now and I intend to carry on for as long as I can.

ZA: What is the problem you are currently facing?

YS: Being a sole trader for 20 years has been hard. I have never expanded my business and have only sold to my local family and friends. This has not benefitted me in the long term. And due to inflation and now Brexit, costs have been at their highest. He has been my accountant for the last 20 years, and he is now planning to retire. He advised me to do my own paperwork from now on. I also get income support, and this is based on how much money I make. To do this, I have had to keep my proof of purchase in the form of receipts, for him to then make the invoices, and it just gets hectic.

ZA: How do you sort out your receipts when handing them over?

- YS: I work from home – as you can see – and I keep my receipts in that box there [points at large rectangular box]. I sort of place my receipts in there and then sort them out by ascending date when it comes close to hand to my accountant. This is a very menial task, and I have often made many mistakes.
- ZA: What type of receipts do you receive when you shop?
- YS: I receive the typical Asda/Tesco's receipt, some A4 receipts for bills, and I sometimes receive handwritten receipts if I go to the market.
- ZA: How do you currently store your important files such as HMC letters and bills?
- YS: I have been using a scanner which I recently purchased. I use this to scan my documents and then I upload them onto Google Drive. I pay about £1.60 a month for 100GB of data storage. It's very possible for me to misplace things, so I have not invested in a physical external hard drive. The files are also on my laptop just in case I needed them and had no access to the internet.
- ZA: That's a great way of storing files. Are you ever worried about the security of them?
- YS: Yes, sometimes. I let people in the house use my laptop if they need it. They download a lot of things and I've heard it's very possible for viruses to attack my files.
- ZA: What problems do you currently face when running your business?
- YS: I find it hard to keep track of the money coming in and out of my business. I find it that I easily misplace receipts/forget to note down a sale, which is poor behaviour on my behalf. I think this may be the cause of my business not doing well in recent years.
- ZA: What do you feel about a desktop application which can help you organise your business?
- YS: Something like this will be ideal!

**-end of extract-**

### Analysis of Interview 1

A main problem Mr. Saleh had with his business is that his sales price had been at a constant for the past 20 years. This can never be good, as he stated, prices of everything else had steadily

been increasing over time. He could have had the opportunity to increase his prices or even turn his business into a partnership with his family/friends, but he decided to keep it the same.

A main issue that he faced was that he found the receipts tracking very hard. The fact that he kept his receipts in 'a box' showed that he didn't put much effort into his organization, and just procrastinated with organizing his receipts as he got them. This would save him a lot of time in the long term as he would not have to sit down and organize them after a long period. Simply a few seconds a day would complete the same job. This would not make the tracking then 'hectic'.

His accountant wanted to retire and advised him to do his own paperwork. To do this manually, it would take a long time. His accountant – as said – is responsible for calculating all his costs using all of Mr Saleh's record of purchases and creating an invoice for his potential income support. This means that Mr Saleh is willing to do all of this himself. This then led me to think that I could create a system which could match my clients' needs. He needs a system which-

- Can input all his sales
- Record all his purchases
- Create a variety of reports to help him track how his business is doing

The interview was a successful one in my opinion. As Mr Saleh works from home, he only had a 15-minute time slot in which I could interview him. This may not have been the best option as he may have been rushing the interview leading to more basic answers. However, I have received enough information to research further into a system I could create for him.

### Current Systems

The essential essence of what Mr Saleh was describing was an accountancy software. Something that would look after his outflows and inflows and would provide a loss sheet. The solution to be created will have to be the most efficient approach to the problem at hand.

Given Mr Saleh's age, it will have to be built with the most intuitive way where there are only methods and usages where they are needed. All the complexities of the program will have to be hidden and written in the background.

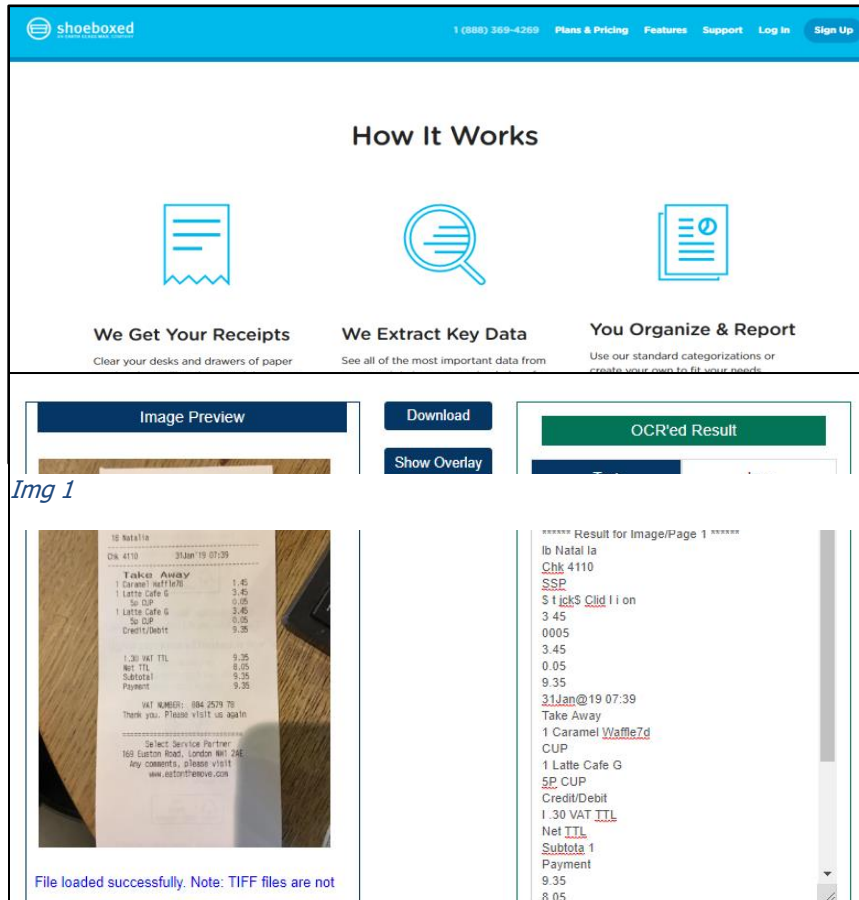
After doing some research I have found some similar systems for sole traders to track the flow of costs and profits using their receipts – which is the main way he tracks sales and costs. Below is the analysis of a few of the systems which could be used by someone like him

#### Shoeboxed and Neat

I have found an application called **"Shoeboxed"**.



Shoeboxed works in a similar way to the system that will be created for Mr. Saleh.

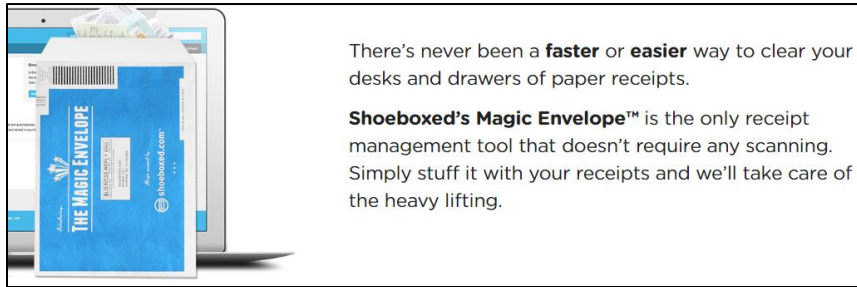


Img 2

As we can see, the system scans receipts and presents the data that is needed in a clear format, where you can also generate reports as needed. It is very easy to use and works efficiently which is perfect for users.

Shoeboxed currently uses an OCR API. An API is an (Application Programming Interface) - a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service. This API works by reading the text of any document it is fed – in the file format of

any image. The details that are extracted are then placed into a .txt file. We can see this in the example below. Here I have used the online OCR API service – this is an example, and the API is available to use in Python. We can feed in the image for a test run. This service is for single reads and cannot be used for a larger scale data scrape. This is an example of the ocr service that may be used by Shoeboxed. There are also many other providers of the service where companies may go to.



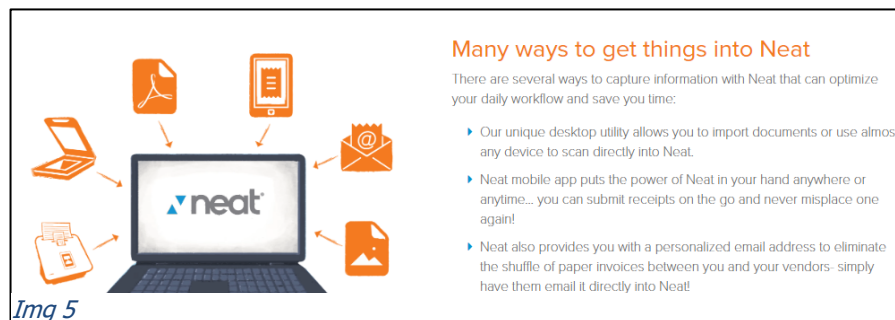
Img 3

This is very limiting however, as for an individual it can cost \$29 (£23.55 in Sept 2019). This would be a monthly cost, and for the price, my client could only get 1 envelope each month. This may not be enough for an individual with a mass number of receipts. It may also take a long time for the service to process the receipts as they must firstly, receive the receipts, then scan masses of them, which would then show up on the system.

It typically takes a few days for post to arrive, and it may not be the best option for many people, due to the time gap.

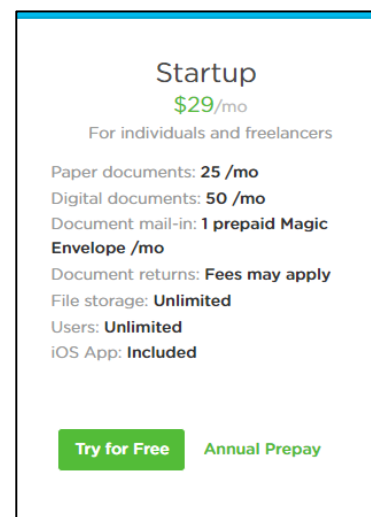
There are many other systems that I have researched, and they all use OCR processing. This means that these businesses use the same API (or a similar version of the API) which allows them to scrape data off receipts.

Another example of a system is **"Neat"**. Neat makes use of the OCR API and allows clients to store their receipts digitally.



Img 5

Shoeboxed also provide a service which allows the customer to send their receipts to them, where they scan all the receipts. This does not use much time of the customer as some customers may have a large mass of receipts.



Img 4

Neat allows users to input their receipts in many ways, including using the mobile phone app and importing documents into their desktop application.

This is useful to the clients as they are not limited to what device they want to use to scan their receipts. Neat also encourages the purchase of a NeatReceiptScanner, which scans receipts to a high quality. OCR processing works perfectly

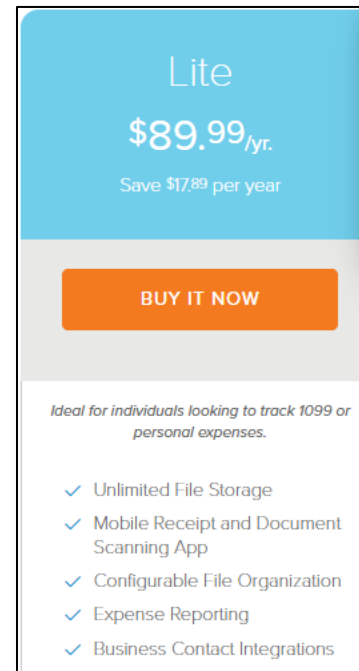
when the image is clean and clear. This device ensures that a clear image has been taken for processing to clearly work.

Neat also makes use of an analysis of all their receipts. They will be able to generate reports according to the needs of their clients, such as Tax Reports, Expense Reports and Spending Tracking.

This is a very good implementation of software that my client may use. Upon researching I found the pricing for the service. The monthly price comes to about £6.00 (on September 14, 2019). This is not a bad price and is quite cheap compared to Shoeboxed. However, a downside of using Neat over Shoeboxed is that you do not get the envelope if you do not want to take images. However, at the price, there is not complaining.

Neat also allows for the user to have an unlimited amount of storage of files, whereas it is limited on Shoeboxed.

Both services are very good when tracking expenses as OCR works well to some extent and both services allow for cloud storage.



Img 6

### Optical Character Recognition

When an image is taken of an object with text, the computer stores it with information such as bit depth, metadata, colour depth etc. It is stored as binary files, specifying the colour and position of each pixel in the image. What we see is the final product; an image. The computer does not **see** the text as we may see it. Optical Character Recognition tries to see the pattern of the pixels to see if it can identify any characters/letters in a group.

There are two options available for OCR, however. Using an API or performing the extraction within python using the Tesseract software engine.

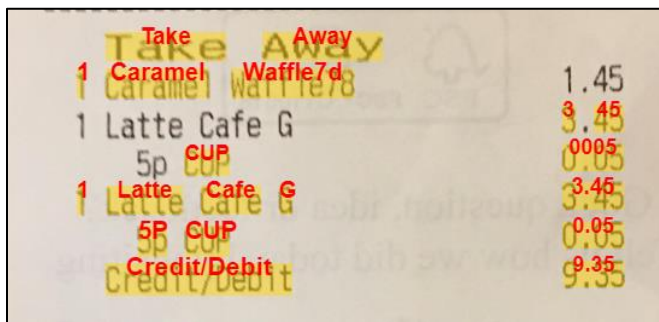
The API (application programming interface) acts as a way for them to solve the problem at hand – here, it is to read and identify the characters in the receipts. This is also a form of abstraction as the complexity has been hidden, but the program is still efficient and does what it needs to do.

Tesseract is a software engine used specifically for OCR. It was created by Hewlett and Packard and is now owned by google. The software is free to use and has been written mostly in C++. This means that the user is not limited to using an online web service and all the data extraction

could be performed without its need. It can be implemented into python and can be manipulated very well. Using the API for an online service is a good choice made by these companies as the service itself is online. If the user is not connected online, they will not be able to use the software. Tesseract OCR would be more sensible to use for a packaged application.

The main selling point of these systems is the tracking of the sales via the OCR. Now, there are many OCR systems which these companies can use to perform the operation, and there is no certain way to know which Shoeboxed and Neat are using.

Now, tesseract and the free API were the only two available to me and are the only ones that will function in my school, so I decided to test these out.



This image shows the overlay extraction of the API that was used. We can see that the price of the first item, and the name of the second product was not extracted. The price of the cup and '5p' was not registered either.

I tried this next on a tesseract prototype I have built in python. Tesseract is different to the API I found online, in that it is available as its own software and does not need to be online. To work in python, I had to install an accompanying module – **'pytesseract'**.

```
Please specify file path
> C:/Users/zubi/Desktop/[project_skyfall]/Programming/GUI/Current/choices/image.jpg
SSP UK
Starbucks Chatham Stat ton
11501451
18 Natalia
Chk 4110 31iJan'19 07:39
Take Away
1 Caramel Waffle7d 1.45
1 Latte Cafe G 3.45
5p CUP 0.05
1 Latte Cafe G 3.45
5p CUP 0.05
Credit/Debit 9.35
1.30 VAT TTL 9.35
Net TTL 8.05

Subtotal 9.39
Payment 9.39
```

I fed the prototype the location of the image, and it performed the extraction. As we can see, Tesseract successfully identified the name of the second product, and the '5p' from earlier. It also provided it in a format

which was more practical as the products and prices in the Online API were given on separate lines. From this we can see that Tesseract is more reliable and has a better chance of working if not 100% accurate.

I played around more with Tesseract and found out that I could output the extraction into a dictionary. This would provide me the location of items in the image, the place they are and most importantly, the text in the dictionary. This would be easy to manipulate.

### Regular expressions

Now came a question of how these services manipulate the data after it has undergone extraction, as there must be a way for these services to output to the user. Upon doing some research I found that using regular expressions may be a suitable way these companies find the data they need. OCR can output a text file where these services may use something like regular expressions to search for data items they need.

Regular expressions are a sequence of patterns used to define search patterns. These patterns are used to identify anything that may fit its scenario. This is very important as the extraction outputs each value as a string, and there will need to be a way to see if there is something like this.

For example, the regular expression for a price will be; `[-+]?([0-9]*\. [0-9]+|[0-9]+)`. This is saying that the string can have zero or more digits, followed by a '.' which is followed by one or more digits. This is a useful way to look for a string inside of a program as we will not be able to tell the program to 'find all prices.'

### Interview 2

I thought that a second interview would be necessary as I needed to discuss the solution I could create. With applications such as Shoeboxed and Neat, the bespoke software should be best suited for his needs. This time, I arranged a meeting for a better suited time over a weekend, when client was free to talk. This is to ensure the interview would not be rushed and he could think about what the best solution could be for him.

#### **-interview begins-**

ZA: So, I wanted to talk to you about the possible solution I could create. Have you ever heard of any service by the names Shoeboxed/Neat?

YS: Never.

ZA: Shoeboxed and Neat are both services which helps business to track their expenses. They do this by taking pictures/scanning your receipts. The cost of items bought are scanned off and added to how much you have spent. Whenever you spend, you scan your receipts. The app scans off necessary data and adds the value to the amount you spend to form a total. Shoeboxed however allows you to send the company an envelope full of receipts in which they do all the hard work. The higher the price paid, the more the receipts you can send. Although for your needs, it does not create a balance sheet for HMC.

- YS: Most of this sounds promising! But the balance sheet is the most important report I will need.
- ZA: Upon doing research, I found a tool which I would be able to optimise which can scan the text of images. Images in your case will be receipts.
- YS: So, you can build a system which does the same?
- ZA: I would be able to build a system which utilises receipts but creates reports which are sensible for your use. What reports do you think would help you organise your business?
- YS: Well for starters, a balance sheet would be the top priority. Would be able to create graphs?
- ZA: Yes. What graphs would you want?
- YS: A sales/time graph, a cost/time graph, a profits graph. These would help me immensely.
- ZA: Not a problem. Would you also like miscellaneous options e.g. days spent more than a value?
- YS: Any tool that would help me, thanks.

**-end of interview-**

Analysis of Interview 2

As we can see from the interview, Mr Saleh needs something more specific for his needs that applications like shoeboxed/neat will not be able to fulfil. This interview in my opinion was also a success as we got to know what he may need for his system to finally complete. We were able to outline the specific reports that my client would find useful and other tools which could help him in any way.

Shoeboxed and Neat would not be able to satisfy him of his needs as he will not be able to create the Balance Sheet which is the most important of the reports. The concept of scanning receipts and letting the program do most of the work will be optimised and innovated to create the reports that will help my client organise his business in the best way he can.

**Proposed System**

The system I will be creating will tackle many issues that are with "Shoeboxed" or "Neat" and make use of free services, limiting Mr. Saleh's costs. The system will be an expenditure tracker and will be able to generate various reports based on the data that is fed into the program via

the OCR system, and will be specialised for his needs. This will be an application that will run on his Windows desktop via Python.

There will be three sections in the program;

- Input Data
- Edit Data
- Generate Report

### Input Data

The main purpose for the program is to track cashflow and provide reports e.g. a balance sheet or how cash is flowing. For the generation of these receipts we need data. And the way we will get data is through the user.

For the reports, there are three main way they will be able to feed data;

- Using Tesseract for receipts
- Manual Receipt Input
- Manual Sales Input

### OCR extraction - Input Data

Firstly, I will be using the Tesseract OCR software. This is because the application will be packaged and may be used offline by my client. This software does also work better with Python. Upon trying out and testing each Tesseract and the API, they both work well, but the API is not very portable in Python. With the API you are also limited with a few requests a day, and you must hope that the API is not down. Tesseract will not have any of this as the software is on the device you are using.

This version of the API is free to use, so does have some limitations when compared to the paid version. From [image 2](#), we can see that the OCR API extracts the data read on the receipt line by line and does not place the text as seen in the receipts. This means that the price of the product and the price of it can be mixed up and jumbled. This would be a problem as the price and item could be mixed up. This would also not be good for my clients tracking of prices. The extraction in image 2 uses engine 1. This engine is supposed to work faster and not be for special characters. Engine 2 however is created for special characters which means they will work better with receipts. Tesseract sometimes does not track all the characters also.

To tackle this, I will be using the inbuilt function within the pytesseract module which will allow me to turn the ocr into a dictionary. Using regular expressions and data manipulation I will be able to find all the data on the receipt as necessary. Regular expressions will help me look for



words in the receipts that are of specific format. For example, if I were looking for a date in format dd/mm/yy, I would write a regular expression to denote that. Similarly, I would do that for the other variables I need for the database. This will allow me to store this in a database from which the reports will be generated.

If, however this still does not work, I will be allowing the user to input their receipts manually. This will mean that the data they enter will be correct and will reduce the chance of error. Sometimes, Tesseract may not completely fail. It may work to some extent by having a few characters off e.g. '£3.90' may be '3&0'. As we can see these are very distinct. And due to their formatting the second value will not be referred to when reports are generated. This can cause a massive problem if the price was significantly more/less.

#### Manual Receipt Input

I will be allowing the user to input their receipts manually if tesseract fails completely/if the receipt they want to enter is handwritten / in a format that tesseract will not understand. This will mean that the data they enter will be correct and will reduce the chance of error. Sometimes, Tesseract may not completely fail. It may work to some extent by having a few characters off e.g. '£3.90' may be '3&0'. As we can see these are very distinct. And due to their formatting the second value will not be referred to when reports are generated. This can cause a massive problem if the price was significantly more/less.

There will be three main fields. Date, Location and Total Price. If the user has specific items they will want to add, they will be able to do so. I believe that these are the most important fields as they will be used to look at the bigger picture. We will be able to generate reports using these values, and it will be the most efficient way.

#### Manual Sales Input

When talking to my client we learn that he takes note of how much he sells a piece of clothing for in a book – he writes it down and adds these up when it is time to create a balance sheet. This is not an efficient way of calculating revenue, as it can a long time to add up 6-month's-worth of revenue. To tackle this, I am planning to allow the user to input their sales as they occur. So, whenever my client has made a sale, they can add this in, and the fields will be stored in the database. This means that my client will not have to make any of the calculations and the cognitive pressure on him to do so will lift.



### Edit Data

Tesseract may sometimes fail, and may work halfway, where only a few characters are off from the desired requirement. This would mean that the user should be given the option to edit the data instead of manually inputting the entire receipts due to a minor mistake.

Here the user will be presented the table of information that is inside of the database. After the user has input some receipt and each receipt has been scanned, the user will be able to see what exactly is inside of the database and will be able to edit it. This will be given to the use inside of the GUI to allow easy inputs, instead of having to write any SQL statements themselves.

This may also be useful if the user has input a wrong character as they will be able to change it. When talking to my client we discussed that the accountant would have to manually add up all the receipts and then check that he has correctly done this, before creating the balance sheet for final. The same case will be with this, except that it will not take as long. This is because all my client will have to do is check the dates and check that total price. He will not have to add up all the values as the program will do that for him.

This will cut down the process quite significantly and will be useful if he needs a general balance sheet to keep a track.

### Generate Reports

Report generation is a key part to the program, and this will be achieved by the generation of multiple reports. The main report that my client desires is the balance sheet as this is the only one that will need to be handed to HMC, but I believe that I need to give him some more flexibility with how his business is doing. He does not have a real measure of how his business does in the short term/how much he is spending/the flow of cash within his business. This is poor organisation on Mr Saleh's behalf, and to tackle this I will be allowing him the flexibility of creating many reports apart from his balance sheet.

### Sales/Time Graph

A Sales/Time Graph will allow him to see how much he is selling over time. This will give him a good scope of how well/how bad is doing. He could use this to make decisions in his business and to visualise how much revenue he is generated. This would be a huge step up from previous terms where he would not have a fair idea of how is doing over time as he would calculate after a 6-month period. Seeing smaller changes may help him make decisions on how much he should sell.

### Costs/Time Graph

A Costs/Time graph will show how much he is spending over time, which he will be able to indicate how much he needs to cut down on his spending/where he can spend a little more. This will also help him visualise his spending habits as he would see how much he is spending at what the period of the graph is like. It may help him decide to spend less and see where he is saving money.

### Profit/Time Graph

A Profit/Time will show him how much he is making – this will allow him to make judgement calls on his small business to see where can invest more for example. Seeing how much profit he is generating over time will help him see the money he has to either invest/save for future expenses. This will help him be aware and influence any decisions that he may make.

### Days Spent on day X

Let's say that the client has had a large shop and has attended many shops and received many receipts. If the user wanted to know how much he spent of that day, he would be able to enter that date. The program would be able to add up all of the total prices of each receipt with that date and provide an output to the user.

### Balance Sheet

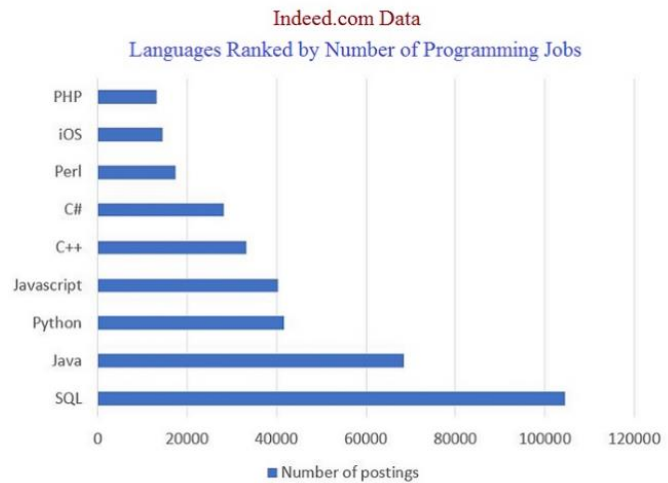
All the graphs however will require a common factor. They will need a date in chronological order. Upon doing some research I saw that I can convert dates as string into its Unix equivalent. This means that it is the seconds from the 1 January 1970 the date given. This will allow me to sort the data appropriately and allow the graph and report data to be accurate.

### SQL Statements

To tackle this, I am planning to use an **SQL** (Search Query Language) database. This is because I would be able to match the names of the products bought along with the price that is attached. I would use the rows and fields to do this. The date will also be able to be seen. This is also because my client wanted to see past expenditures in the form of a table so that he knows what he is spending his money on, and how he can track back.

I have chosen SQL over a Panda dataframe as Pandas are generally used for data analysis rather than a simple database. The database will allow me to INSERT, SELECT, UPDATE as instead of writing in pure python. I will be importing the MySQL module for python so I will not have to translate the SQL code after writing it out separately.

SQL has also become a very popular language to use over the last 40+ years it has existed. (See graph)



The data that would exist on the SQL database would be very sensitive, and it could be susceptible to attack. To stop this from happening I should create limits to stop SQL injections.

### SQL Statements

To handle the different data I will be using, I will be using a database. SQL is a DBMS (Database Management System) that allows me to interact with the database created. Using SQL means that only the programmer (me) will be able to access the database, as all the complexities will need to be understood by me when it is created.

As I will be using Python to write this program, I will be using the '**sqlite3**' module, as this is the module available to us at our school and is a preinstalled package in Python 3. There are a series of commands which are available to use in sqlite3. These can be split into three sections:

- DDL (Data Definition Language)
  1. CREATE – used to create a new table in an SQLite database
  2. ALTER – used to rename a column, table or to add a column
  3. DROP – remove a table added with the CREATE command
- DML (Data Manipulation Language)
  1. INSERT INTO ...
    - a. VALUES – creates one or more new rows in an existing table
    - b. DEFAULT VALUES – inserts a single new row into the table
  2. UPDATE – used to modify a subset of the values stored in zero/more rows of the database.
  3. DELETE – removes records from the table
  4. WHERE – only rows that meet specified criteria are affected

- DQL (Data Query Language)
  1. SELECT – used to query the database

All the above will be used to create the database to store all the necessary details for my client to create the invoices. The need for a database rather than a Panda DataFrame is that Pandas are typically used for data manipulation and are set out in tabular form including both strings and numbers. SQL allows me to form a relationship between the different tables that will be needed. This makes SQL better for my use, instead of creating multiple DataFrames with no inherent relationship.

I will need a variety of tables to use when creating my database. For this I will need to use the CREATE statement to initialize my tables. Once all the tables are made, I will need to add data from the receipts – this is achieved by the INSERT INTO VALUES statement where I can insert values into the headings specified. I will be able to insert values such as 'dates, price and item name.

I can use the various statements to achieve;

- CREATE – to initialise tables in my database
- INSERT INTO VALUES – to insert values (e.g. ocr'd data from the receipt) into the accompanying columns
- SELECT – to select certain values for comparisons of dates/values.

### Different Receipt Types

Secondly, this system will not be limited to one type of receipt like the services that we saw. Receipts take many forms and they can be interpreted in different ways. From images 7 & 8, we can see 2 examples of a non-typical paper receipt. All the receipts however have the same thing in common. They all have the name of the item bought, the price, and the date of purchase. This is the key information that my client needs.

Zahlungspositionen und Preis				
Positionen	Preis	MwSt (D) 19%	MwSt D: 7%	
ICE Fahrkarte	1 87,00€	87,00€	13,89€	
Reservierung	1 0,00€			
Zahlungsmittelentgelt	1 0,75€	0,75€	0,12€	
Summe	87,75€	87,75€	14,01€	
Kreditkartenzahlung				
Betrag	87,75€	VU-Nr	455665619	Transaktions-Nr 680613
Datum	22.10.2017	Gen-Nr	NINF8LC	
Ihre Kreditkarte wurde mit dem oben genannten Betrag belastet. Die Buchung Ihres Online-Tickets erfolgte am 22.10.2017 16:27 Uhr. DB Fernverkehr AG/DB Regio AG, Stephensonstr. 1, 60326 Frankfurt, Steuernummer: 29/001/60002.				
Ihre Reiseverbindung und Reservierung Hinfahrt am 22.10.2017				
Halt	Datum	Zeit	Gleis	Produkte
Leipzig Hbf	22.10.	ab 18:15	10	ICE 1206
Hamburg Hbf	22.10.	an 21:24	6	
1 Sitzplatz, Wg. 38, Pl. 66, 1 Fenster, Abteil, Nichtraucher, Res.Nr. 8081 0010 2390 15				

*Img 7*

To read this key data from different types of receipts, I will have to refine the OCR API to my needs. The algorithm I create should know what it is specifically looking for, and how to put it in a key format into the database.

As was identified by my client in the second interview, some receipts have been written out by hand. This would be quite difficult for

the API to read as every person on the planet has different handwriting. To overcome this, an input box will be provided on the program for any receipts that were written by hand. This would be the best way to do so, however there are a few disadvantages:

- Human error could mean that a price is entered wrong
- The input box could be exploited to show a higher/lower price for the sake of a higher income support
- A hacker could tamper with the data

### Adaptive Binarization with High Threshold

This essentially means that the white colours and black colours will be put up to the highest contrasts. This will allow the OCR system to fully recognise each character on the page.

If this method also fails, then the user will have the option to see the data that has been scraped from the receipt and fix any errors that have occurred. There is a final fail-safe option where the user can input the data manually – this will store the necessary data and should only be used if the OCR fails completely. This method may be used if the receipt is written by hand, which the OCR may not actually recognise.

### Loss Sheet

Work done	£	p
Accountancy	290	
Materials	240	
Power, light and heat	411	
Sundries	54	
Telephone	150	
Travelling	85	
	1,140	
NET PROFIT FOR THE YEAR	2,710	

NOTE:  
One third of light and heat, and one half of telephone expenses have been charged to the Profit and Loss Account.  
I approve the above accounts and confirm that I have supplied all the relevant information and explanation in their preparation.  
In accordance with the instruction, information and records supplied to us we have prepared the above Profit and Loss Account and certify the same to be in accordance therewith.

Here is the loss sheet that was initially created by the accountant – and this is the document that will need to be generated with the data that we will have.

The variable 'Work Done' is equal to the revenue of the client. Through the program this will have to be calculated by adding up the revenue from the dates given by the user for the report.

The 'Expenses' variable is split into many parts – Materials, Bills, Telephone and Travel. Accountancy will not be needed in the new loss account, as the program will create the document. These variables will also be created by using the database and adding up each value that is held within in. Each type of expense will be given a label when added up so they can be placed accordingly in the document. Total

costs will be the result of the sum of the expenses.

Finally, the net profit will be the result of the subtraction of the revenue from the total costs. This will also be placed into the document.

To hand in this document, Mr Saleh must sign and date it before handing it in.

## End User System

There is a specific way the user will use the program, but before they do use the program, there is a specific set of requirements that will need to be met. The user must;

- Have a desktop PC which can run the Windows OS – along with necessary hardware
- Can use a file directory on the PC
- A device which can take high quality images of receipts
- A USB to transfer images from camera device (DSLR, Smartphone) to PC
- Have images (.jpg, .png, etc) of the documents
- Have these files in a directory which can be accessed
- Have a PC which can run the Tesseract engine
- Have a PC which can run Python3.x
- 

Given that the user has access to all of these, we can see the flowcharts in the modelling section below the objectives.

## Smart Objectives

There are many things that I will need to achieve when completing this project. This is to ensure that the system runs as smoothly as it can. The following are a range of objectives that will have to be met in order to run the smoothest it can. The program must meet each and every objective in order to meet the client's needs.

Objective		
1	(a)	Program must initialise when clicked on main program.
	(b)	Log In page must be displayed with input boxes for username and password.
	(c)	Button to submit username and password must also be displayed
2	(a)	User must be able to input values in the input boxes
	(b)	Must be able to click on the submit button
	(c)	When button is clicked, the program should access a database with the username and password.
	(d)	If there is a match, program should let that user in. e.g. if client 1 enters, they would not be able to access client 2's records.

	(e)	If there is no match, program should allow user to input values again and give an error message
3	(a)	If access is granted, the program should display three buttons; Input, Edit, Reports
	(b)	If the user clicks on "Input", the input page should display
	(c)	If the user clicks on "Edit", the edit page should display
	(d)	If the user clicks on "Reports", the reports page should display
4	(a)	If the user chooses Input, this page should display with the table, and necessary buttons
	(b)	i If the user chooses to use the Tesseract Scanner, they should be able to choose the image files of the receipts they want to scan
		ii The table should be able to store the names of the files when they are selected
		iii The table should store file types that are supported by Tesseract e.g. png, jpg
		iv If the wrong file type is selected, then this must be removed, and a message should display
		v Assuming all files have been selected, the user should be able to click on the "scan" button
		vi If the table has no images and the "scan" button is clicked, then an error message should display
		vii Tesseract should store all OCR'd content into a dictionary
		viii If OCR fails, then an error message should display
		ix Using the Regular expressions, the key values needed from receipts should be scraped and assigned to a variable
		x The dates should be located and converted to its unix equivalent as mentioned above
		xi These variables should store in the database with allocated fields
		xii A success message should be given after every successful scan
	(c)	i If the user chooses to input their receipts manually, they should be able to click on a button
		ii The button should display three input boxes which are the most important fields for reports generation; dates, total price, location shopped at
		iii The user should be able to input values in the boxes
		iv The dates must be in format "dd/mm/yyyy", total price must be a float, and location must not be empty
		v If the above validations are not met when the submit button is clicked, an error must be given
		vi If the user wants to add specific items from the receipts, then they can add each item with allocated fields.
		vii If all validations are met, then the program must put all values into the database in the allocated columns
	(d)	i If the user wants to input their sales made, then they should be able to click on a button
		ii This button should display a menu with fields the user can enter data along with a submit button
		iii The user should be able to input a date, item sold, and price sold at.

		iv	If these are not in the correct format, when the submit button is clicked then then an error message must be displayed
		v	If all validations have been met, then these values should be placed on the database along with a calculated unix date.
5	(a)	i	The program should display the tables using the fields related to the clients account. E.g. Client One should see all their details.
		ii	The tables must be updated to the latest data if data has just been input
		iii	Two tables must be displayed: one for receipts and one for items
	(b)	i	The receipts table and items table should have the same fields as the inputted data
		ii	Each record must be able to be selected

Objective Number	
(5)	Editing Data; The user must be able to edit the tables that are stored in the database; these are the receipt and item tables. The data in the GUI must be brought from the database and as they are updated/deleted, the values in the database should change accordingly. This must be done in an efficient manner.
(6)	Reports Generation; The user must be able to generate reports according to the one that they choose. If the user chooses to generate a graph, then the graph must be generated based on the dates that are given. This should pop up and allow the user to interact with it.



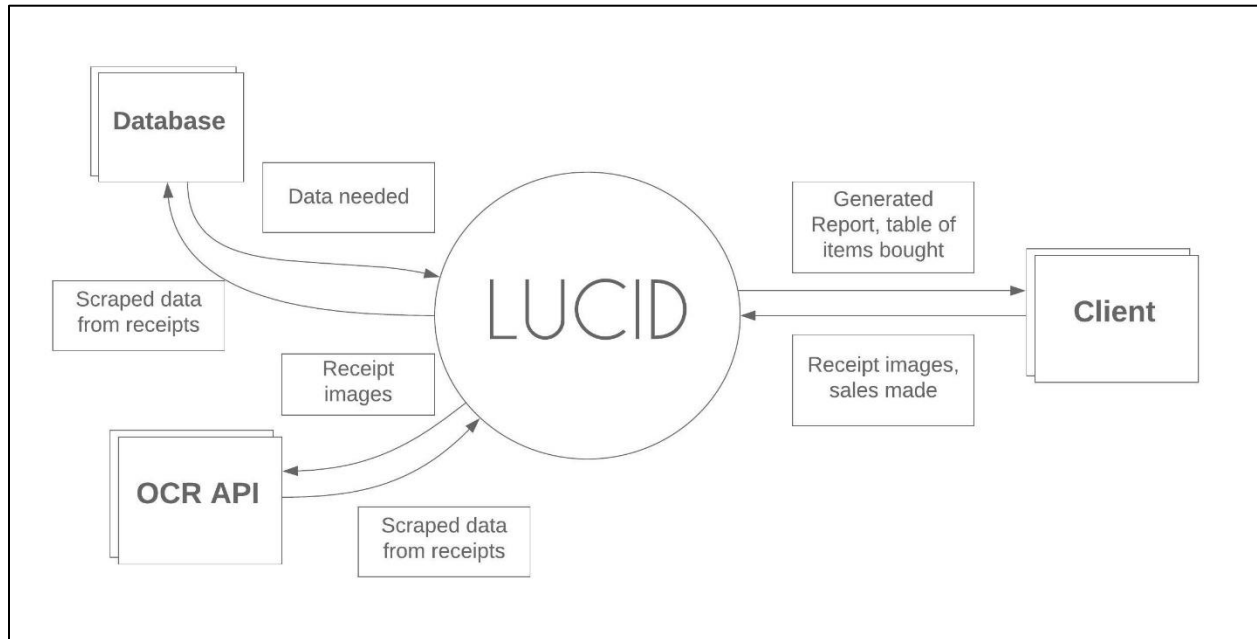


## Modelling the solution

The solution will need many diagrams to show how the data will flow between the program.  
From the user's inputs to the outputs given.

## Data Flow Diagrams

### Level 0

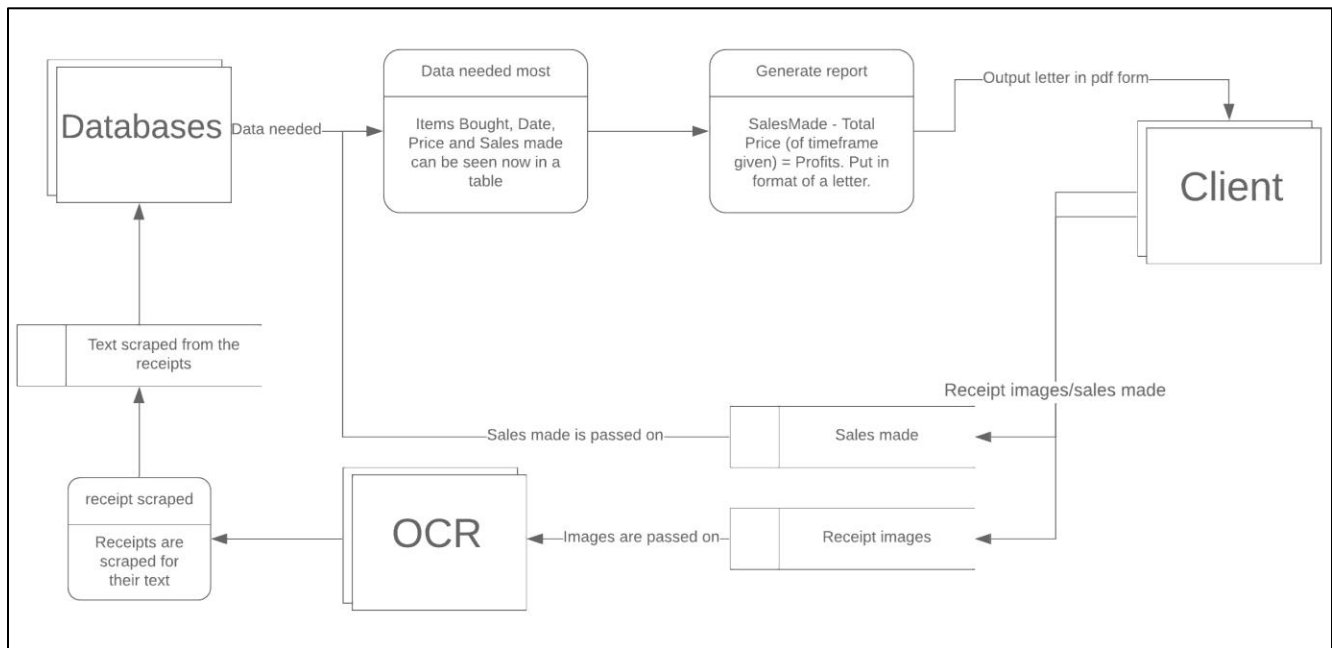


Here we can see the flow of data from the system I will create – Lucid – to the different entities that exist. The Client will input data such as the receipt images, sales made or manual data/corrections to the data frame. This will then be passed on to the system.

The images will be passed on to the OCR system where the extraction will take place. This data will be handed back to the system.

The same data will be passed on to the database. Here the database will handle all the necessary data and will allow access for the system where the reports can be generated using the data. This will also be able to be seen by the user, where they can make any changes to wrong extraction by the OCR system.

Level 1

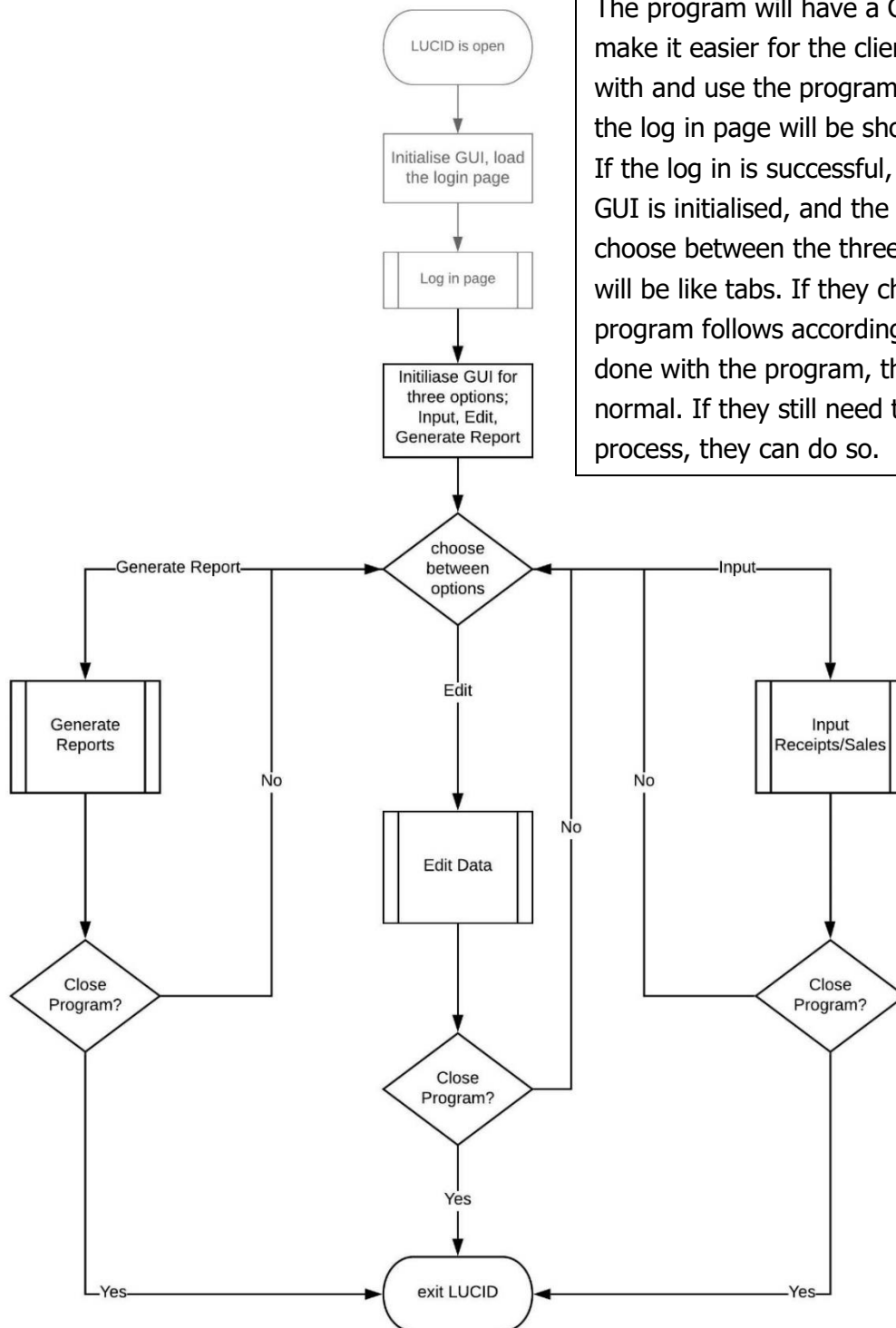


Here we can see how the different entities will interact with each other. The flow of data begins with the client and ends with the client. They initially can enter two things; receipt images and sales made. The receipts are passed onto the OCR system where the receipts are scanned for the text that the algorithms detect. This is then stored.

This is then passed on to the database where all the text is sorted out from being in a poor format to a tabular form where you can initialise the database and normalise the data for what is needed. The necessary data e.g. price of item, name, and date purchased will all be passed on to the main system. These will be seen alongside the sales made by the user which they will input manually. Calculations will be made accordingly, and the report will be generated using the data given to us by the user in the necessary format as they will require it to be in.

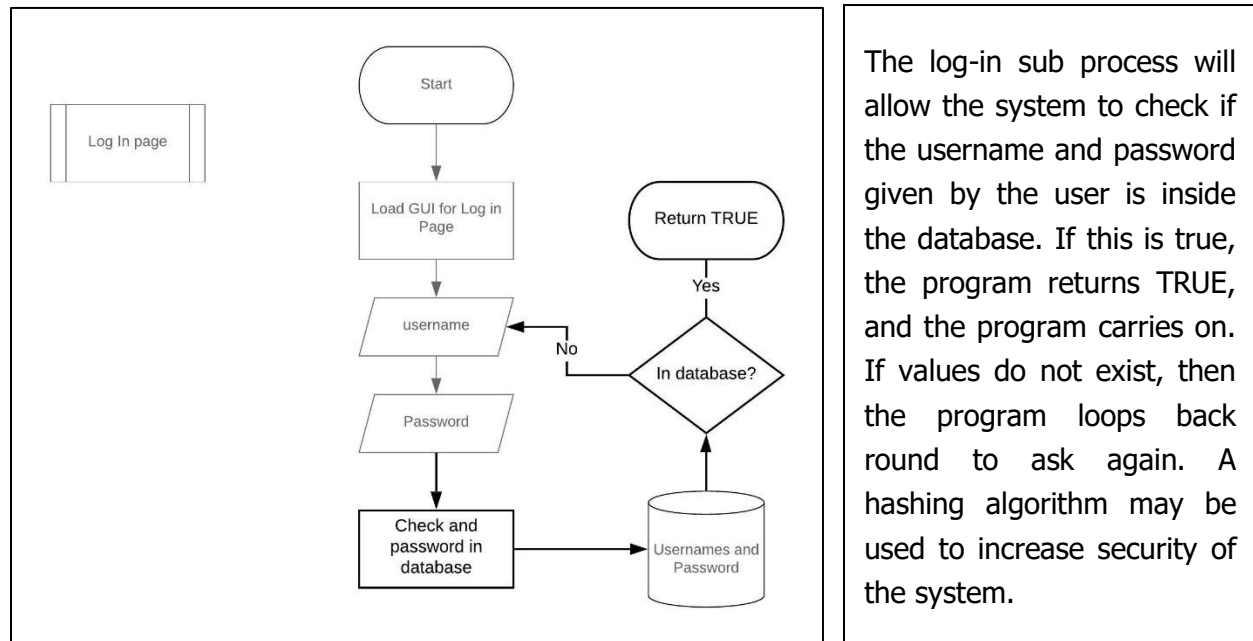
## System Flowchart

### Final Flowchart

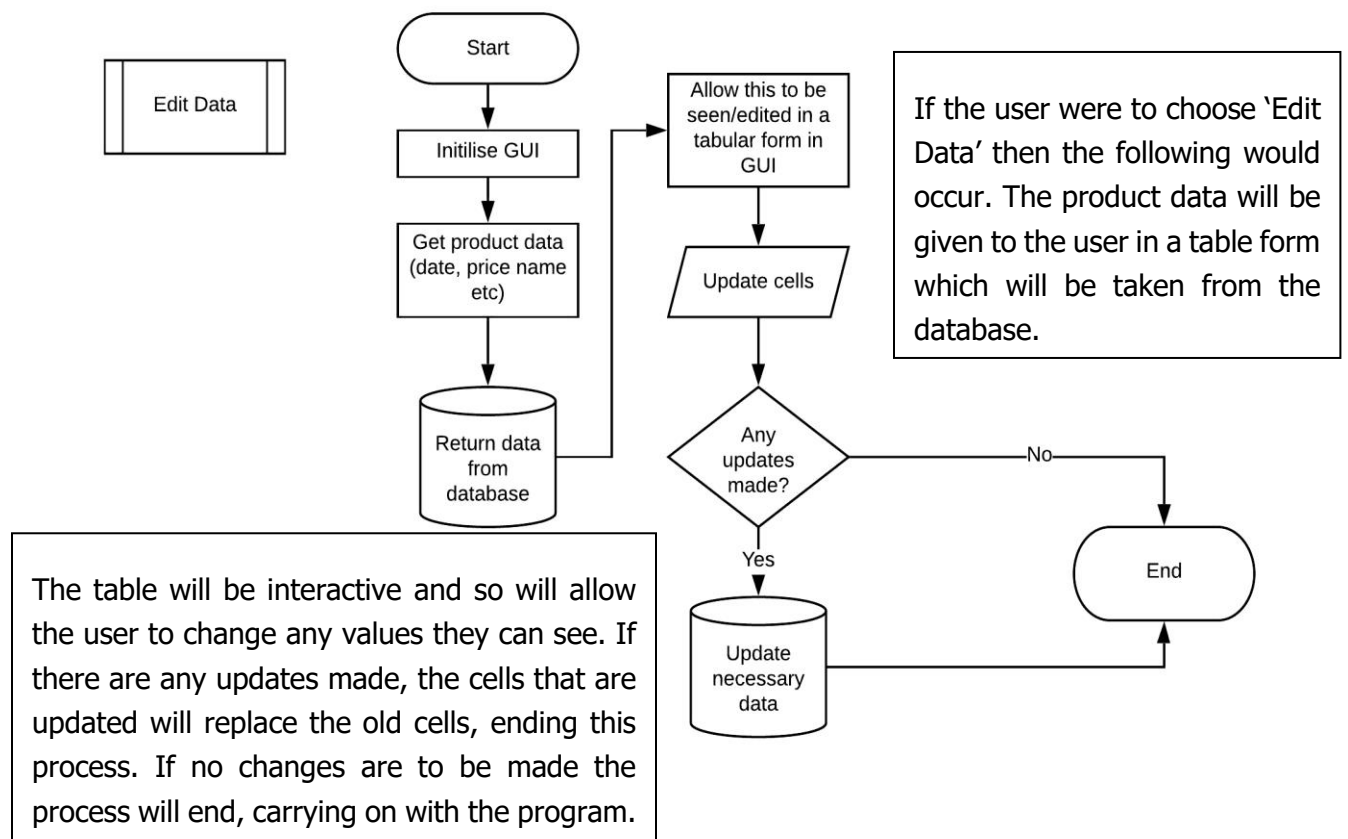


The program will have a GUI which will make it easier for the client to interact with and use the program. From here, the log in page will be shown to the user. If the log in is successful, the rest of the GUI is initialised, and the user can choose between the three options. These will be like tabs. If they choose any, the program follows accordingly. If they are done with the program, they can close as normal. If they still need to use another process, they can do so.

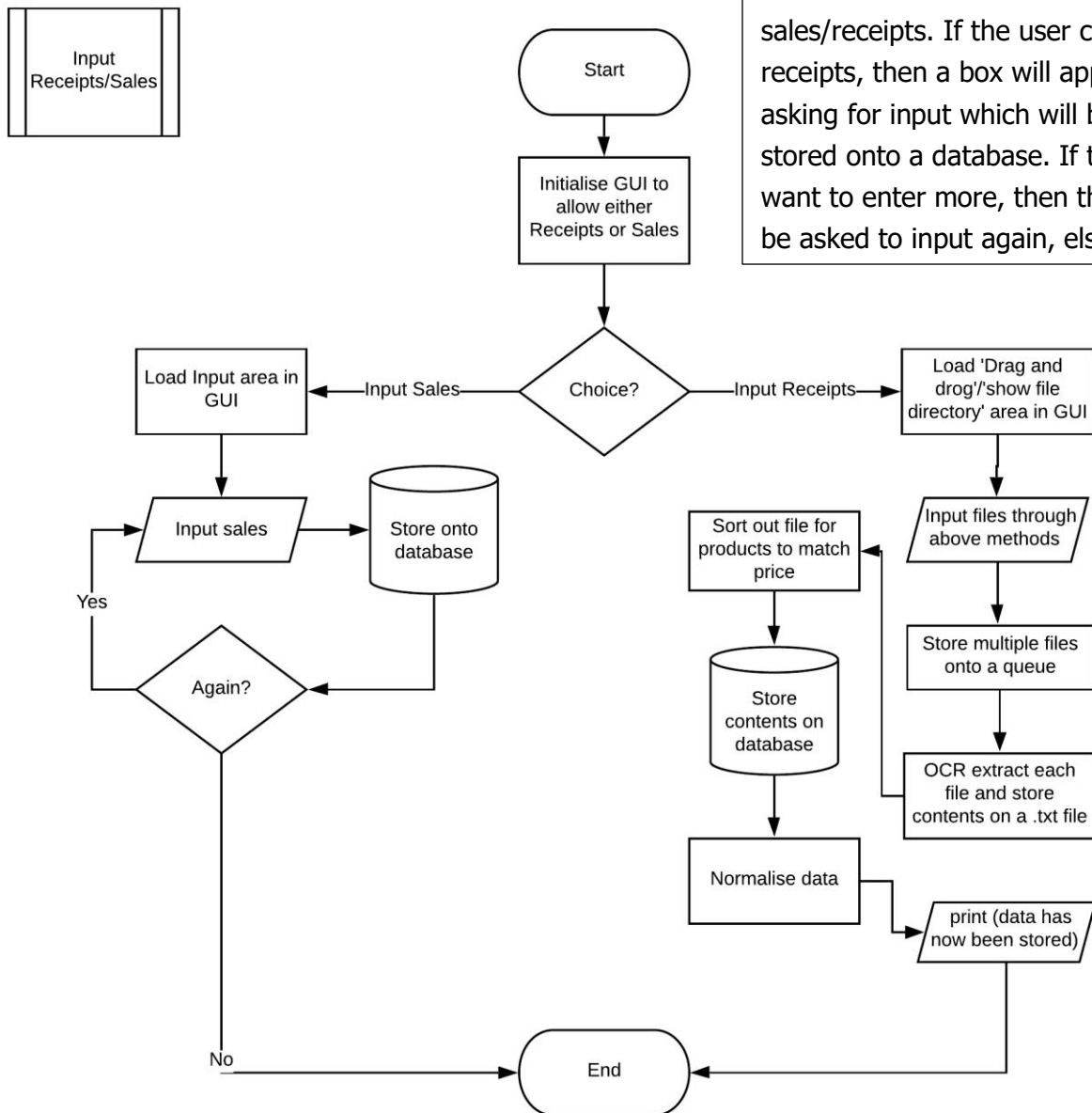
## Log In



## Edit Data



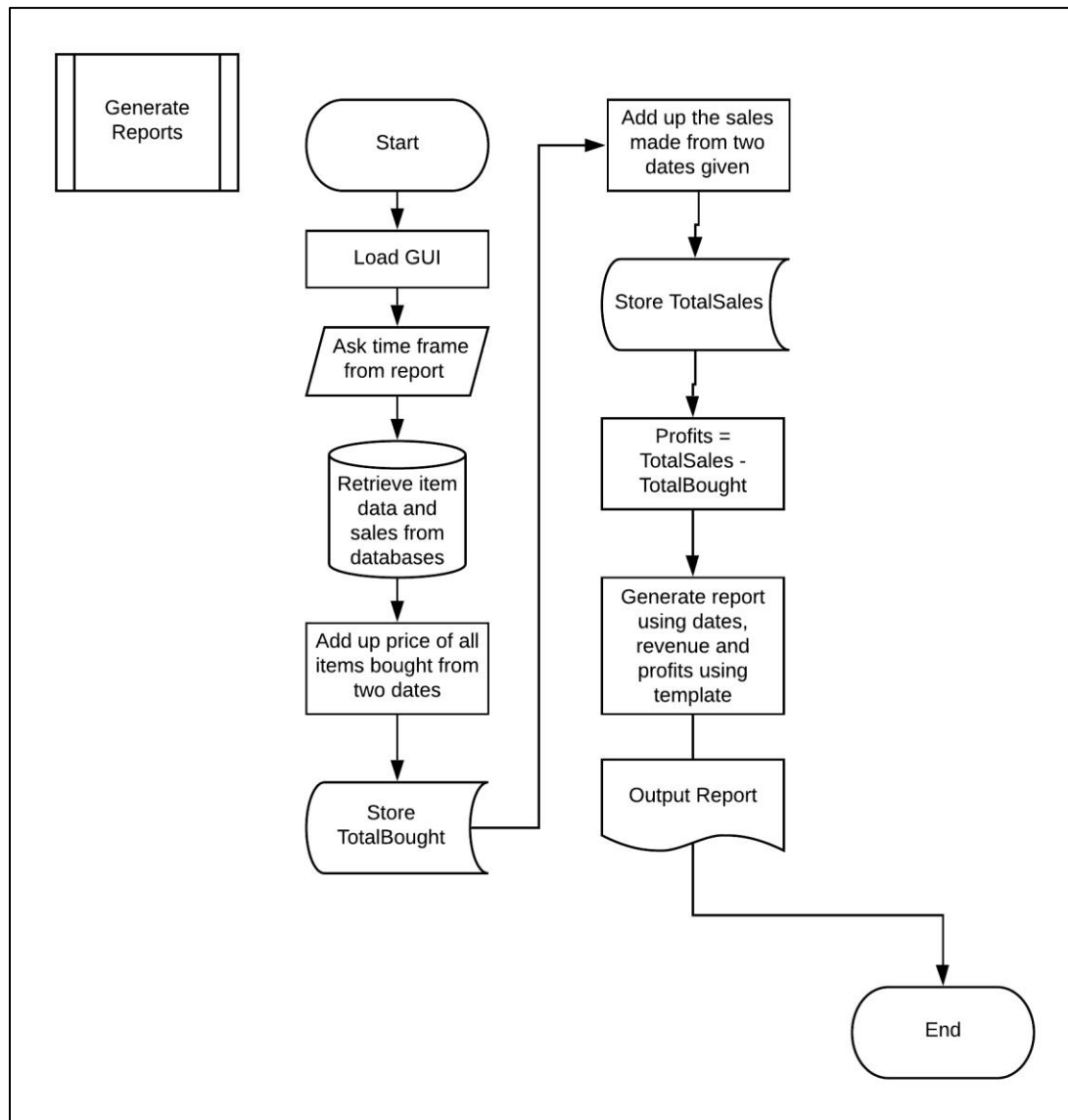
## Input Receipt/Sales



This process is begun, and the user can choose whether to input sales/receipts. If the user chooses receipts, then a box will appear asking for input which will be stored onto a database. If they want to enter more, then they will be asked to input again, else end.

If the user was to choose receipts, then an option to choose path or 'drag and drop files' will be given. These files will be put onto a queue and one by one they will be extracted using OCR whose contents will be placed into a .txt file. As we saw earlier, the price and product names are sometimes not placed next to each other. This will be processed for them to do so. This data will then be stored on a database where it will be normalised. The user will be known that the data has been stored and the program will continue.

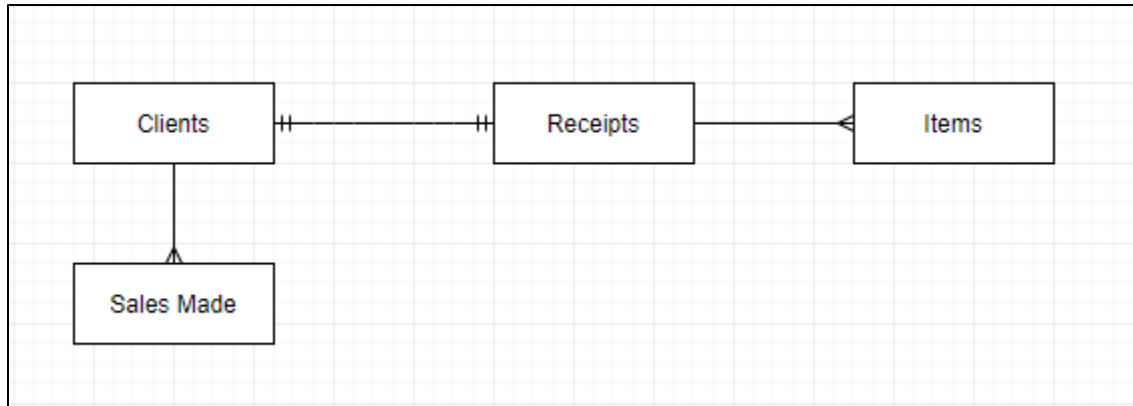
## Generate Reports



If the user chooses to generate the reports, the following will occur. The GUI will be loaded, and a time frame will be asked. This will be cross referenced by the dates in the databases. The price of everything bought will be added up and stored to a variable. The total revenue will be added and stored to another variable. A calculation will be made, and the report will be made using a template that will be created beforehand. The dates, revenue and profit calculated will fill in the blanks and the document will be created. This will either be in a pdf format or a word processing file format. The program will then continue as normal.

## ERD

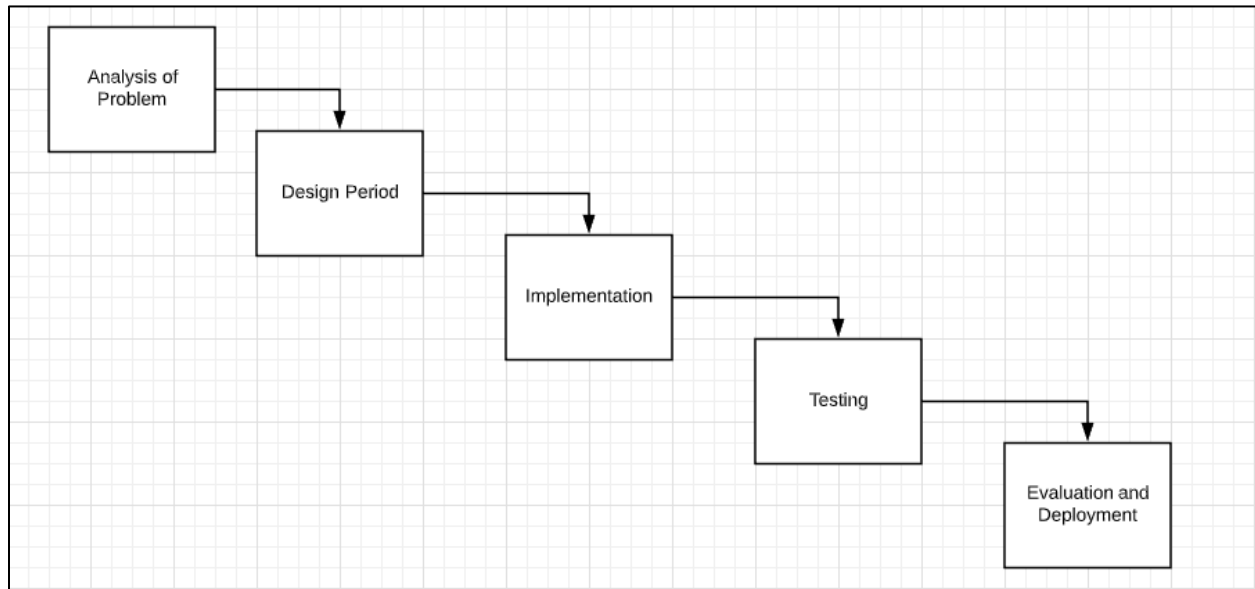
An ERD is an entity relationship diagram. It describes the relationship between the different entities and how they interact with each other.



- My **client** has many **receipts**, but they are all linked to him. All the entities have the same person involved – this is therefore a one-to-one relationship.
- One **Receipt** may have many **Items** – this is therefore a one-to-many relationship.
- My client has many sales and he can sell many things. This means that the **Sales Made** and **Clients** will have a one-to-many relationship as all these entities will have one client.



## Project Approach



For the project approach I will be using the Waterfall method. I have opted for the waterfall method because I believe it is the most relevant to my problem at hand. It is the earliest method of breaking down a computing problem and steps to ensure the problem has been solved. Steps in this method are sequential meaning that they can happen one after the other. Steps do not overlap meaning that once one has finished, you can move onto the next. However, the system would be developed iteratively, ensuring it is the best system that can be made.

I have decided to choose this method over Agile, as my system will be unchanging. Typically, Agile is used to solve a problem and then make the solution better and better. For software developers to innovate due to the changing market. My client will need this software for one purpose, and due to this, it will not need innovating. This may be a disadvantage however as systems such as Neat and Shoeboxed may advance significantly over the years, making my system obsolete.

**Analysis of the Problem** – allows for all the possible requirements for the system that is being proposed. Conducting research, talking to my client, creating initial flow diagrams to show how the system may work etc. This is the initial beginning to the problem.

The next step to creating the solution is the **System Design**. Here, we take the requirements that are made in the Analysis, and the system design is prepared. It can help in identifying and specifying hardware and system requirements which can help the system run. The flow diagrams can be further developed, and a pseudocode can be created.

**Implementation** is a form of decomposition. This phase allows for the problem to be broken down into smaller units. These smaller units will be developed and programmed for their own functionality and then tested as their own individual units – this is called unit testing. Here we can also put all the units together to form the solution at hand.

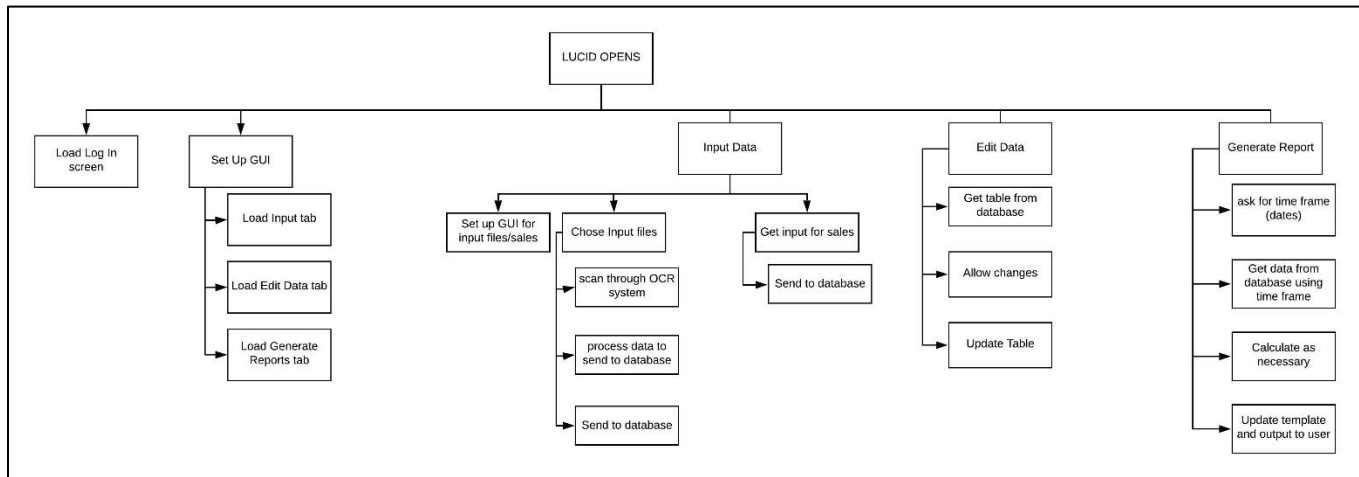
Once the program is put together, we put it into the **testing** phase. Here we test for each input and output, the different units work together, and make sure that the program can perform up to the standard of the proposed solution. Once the program is tested, final tweaks are made, and the program is ready to be used.

In the final stage, the solution will be **evaluated**. This means that I will have to discuss what worked in my project and what did not, and how I would go about it if I were to go about a similar project. The system will also be available for the Mr. Saleh to use. He will also tell me how he is finding the system and what he would prefer different from a client's perspective.

## Documented Design

### Overall System Design

#### Top Down Chart



Above is a top down chart explaining how the different parts of the program will need to be created and it the many steps. This is a method of implementation and decomposition. Breaking down the larger end goal – Lucid – into smaller chunks. Each element of the chart will need to be programmed and put together for the final program to work.

Although the three tabs, Input Data, Edit Data, and Generate Report have been put in that order, the user should be able to choose between them as indicated in the flowcharts. It will be up to them to use whichever one they choose. There will be many steps in completing this.

#### Data Dictionary

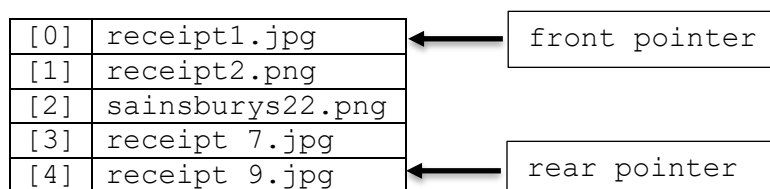
Data Item	Data Type	Validation	Sample Data
<b>Username</b>	string	Match value in database	ZXarif21
<b>Password</b>	string	Match value in database	B3nAs9*uI
<b>Dates of sale</b>	date	isDate	21/09/18
<b>Product names</b>	string	NOTNULL	Velvet Dress
<b>Product Price</b>	string	NOTNULL	£8.99
<b>Item bought price</b>	string	NOTNULL	£5.49
<b>Item Name</b>	string	NOTNULL	Chicken Fillet

<b>Item bought date</b>	date	isDate	12/03/14
<b>Tab being used</b>	boolean		InputFiles = True EditFiles = False
<b>Table being edited</b>	2D-Array		[["yellow hat", "£3.50"],["Green Socks", "3.20"]]

## Data Structures

### Queues

As seen from above, I am planning to use a queue to track the scanned receipts one by one. The scanning will occur regularly, and the client may select a large group of receipts in one go. To scan each one, I will be using a queue. The receipts will be in an image format e.g. .jpg, .png etc. This will allow the OCR system to work well with it. Below is an example of how it may work.

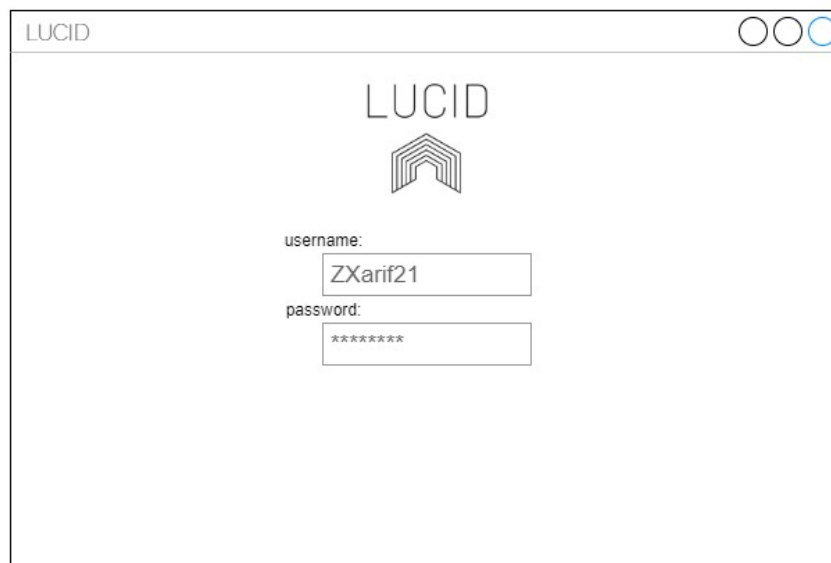


Here we can see that a queue has been created with a front and rear pointer. Front pointer is at position 0, and the rear pointer is at position 4. This means that the queue I will be implementing will be a static data structure as the memory locations will already be allocated and will not have to go to the heap to allocate memory locations. If there is an overflow of data, the program will wait for a free space and fill this one up. This also means that this will be a circular queue so empty locations are not filled. If I were to use a dynamic data structure, the program will have to allocate memory from the heap, which will be a waste of time. Using a linear queue would be a waste of memory as empty locations will not be used. Using a circular queue will mean that I will need variables which will keep track of the front and rear pointers.

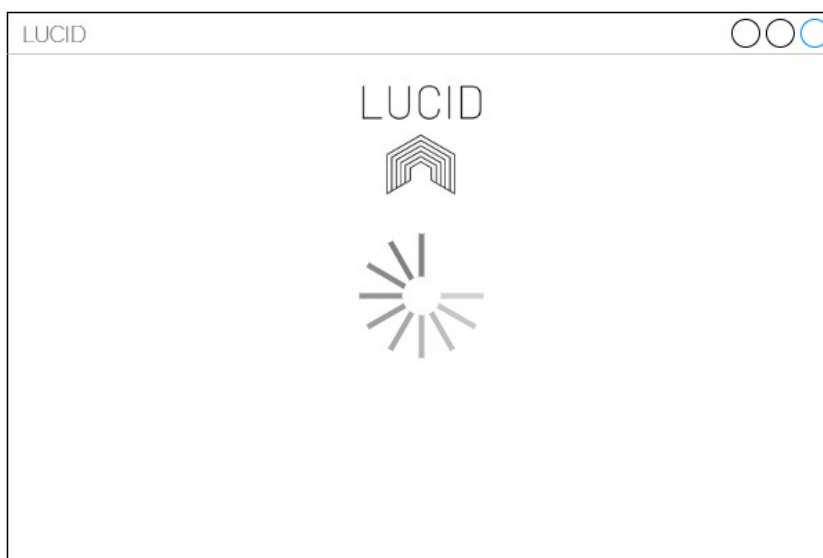
## User Interface Design

Below is a **prototype** of how the user interface could look when run on a Microsoft Windows platform as discussed. Each section will show how the end-product will look to the client when he is using it. This GUI is a key part of the product requirement as it should be easy for him to use without any complications.

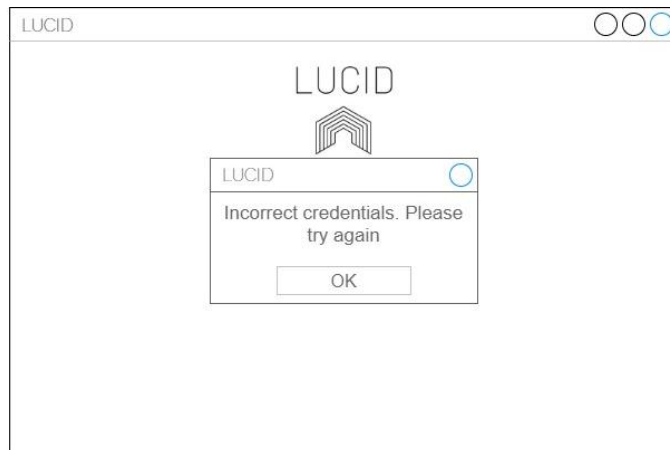
### Log-In System



When the program is run, the user will be welcomed by the log in screen. They can enter the username and password which will be given to them when they sign up for the program. In the case of my client, he will be given the credentials when the program is finished. I have decided to keep the design very simple, so it is easier for my client to navigate.



Once they log in, they can press enter, and the program checks if the credentials are inside the database. If they are, the program carries on to the rest of the program. If they are not, the program returns a pop-up box which tells the user to try again.



This is given to the user so that they can try again. This is to maintain security as the data that will be held will be confidential.

```
INITILIZE USER-INTERFACE
INPUT username
INPUT password

[SELECT * FROM Client WHERE Username = ? AND
Password = ?]
IF username in Username AND password in Password
    THEN SUB mainProgram()
ELSE
    Display Error Box
ENDIF
```

Here, the log in GUI is initialised and the input boxes are shown. The user can input their username and password. The program checks if these values are inside the database, and if they are, the program

```
sub hashAlgorithm()
    FOR each char in password:
        Take Binary equivalent
        perform XOR on the addition
        Replace every 0 with ",£,$
```

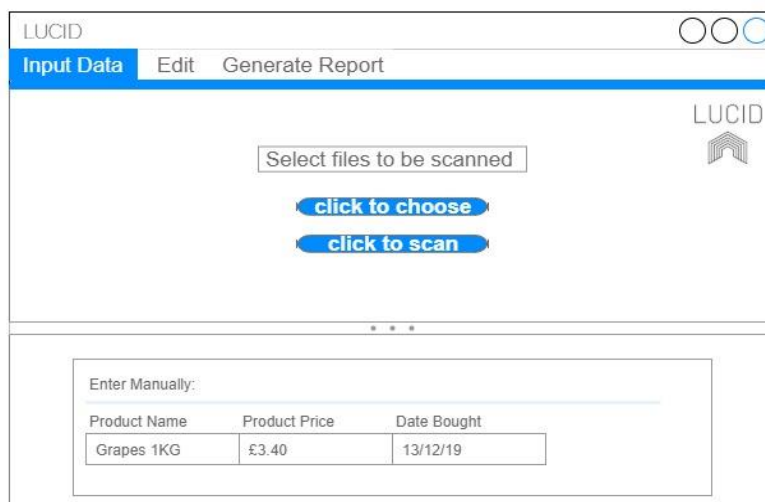
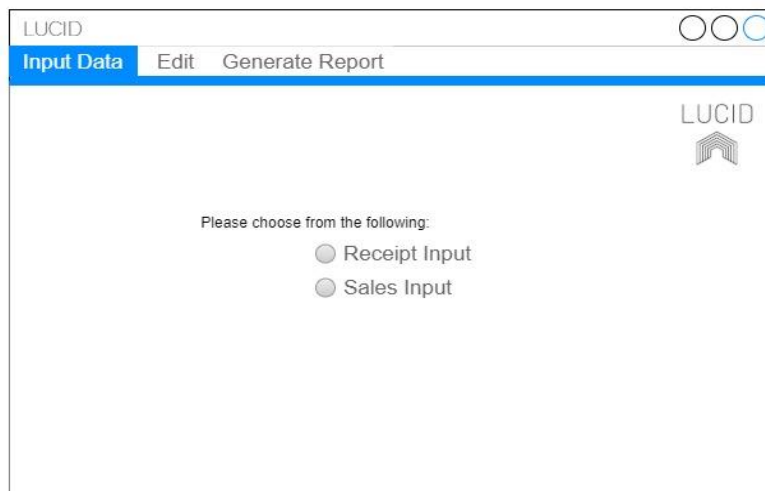
continues. If not, an error box is shown, allowing the user to input their details again.

## Main Program

```
Def SUB mainProgram()  
    choice1, choice2, choice3 = False  
    choice = input("Click a button")  
    IF choice == choice1  
        SUB inputData()  
    ELSEIF choice == choice2  
        SUB editData()  
    ELSEIF choice == choice3  
        SUB generateReport()
```

The user will get to choose which process they would like to complete given that they are able to log in. They will be able to choose between 3 options. Input data, edit data, and generate a report. Below are examples of pseudocode of how they will come to be.

## Input Receipts and Sales



Product Name	Product Price	Date Bought
Grapes 1KG	£3.40	13/12/19

If the log in details are correct, this screen is given to the user with three distinct tabs. They can choose which one they want to use. If they want to input data, they can click the tab and choose the following options.

If they choose receipt input, they will be greeted with the screen as shown. If they want to scan the receipt, they can click the button where it will allow you to specify the file location of the files. Once this has been selected, they can click to scan. The files will be put onto a queue and scanned. If they choose to enter manually, they can enter the data in the fields as shown in the example. This is more for handwritten receipts as the system will not support this.

## Input Sales

Input Data is split up into two sections. Input sales and input costs. Costs can be input in two different ways; through the tesseraact OCR or manual input for handwritten receipts. Sales are to be input manually as these are also recorded by hand.

```
def SUB inputReceiptData():
    date = input()
    location = input()
    totalPrice = input()
    if choice == item:
        receipt_id = SQL(SELECT maxsize(receipt_id)
                           FROM RECEIPTS)
        for i in range(items):
            SQL(INSERT INTO items
                values,
                itemName = name, itemPrice = Price)

def SUB scanImage()
    DisplayButton = False
    InputManually = False
    //The button may be clicked by the user
    IF DisplayButton = True THEN
        OPEN FileExplorer
        ReceiptFolder = path chosen by user
        queueReceipt = MAXSIZE[5]
        overflow = []
        front = 0
        end = 1
        FOR each image in ReceiptFolder
            queueReceipt = enqueue(image)
            end += 1
            IF front == end:
                overflow.append(image)
        SUB OCRImageScrape(queueReceipt, overflow)
    //User may choose to input manually
    ELSEIF InputManually = True
        ItemPrice = input("> ")
        ItemName = input("> ")
        DatePurchased = input("> ")
        IF ItemPrice, ItemName, DatePurchased != NULL THEN
            INSERT INTO Receipts(itemsPurchased, itemsPrice,
DateOfPurchase)
            VALUES (ItemName, ItemPrice, DatePurchased)
```



```
def SUB OCRImageScrape(queueReceipt, overflow)
    //get tesseract
    //before scraping we need images in .png and pre-process them

    FOR EACH image in queueReceipt and overflow
        IF image[:4] != ".png"
            image = image - image[:4] + ".png"
    FOR EACH image in queueReceipt
        grey = threshold(image, grey) //creates contrast
        //between black and white characters
        text = tesseract(image)
        f = open (image, "a")
        f.write(text)
        f.close
        dequeue(image)
        front += 1
        enqueue(overflow)
        end +=1
    UNTIL overflow == NULL
    FOR each file
        f = open (file, "r")
        binaryTextSearch(file, date) // gets date
        binaryTextSearch(file, price) // gets price
        binaryTextSearch(file, itemNames) //gets item names
    IF itemNames, price, date != NULL THEN
        INSERT INTO Receipts(itemsPurchased, itemsPrice,
DateOfPurchase)
            VALUES(ItemName, ItemPrice, DatePurchased)
    ELSE
        DisplayErrorMessage("Receipt Scanning Failed")

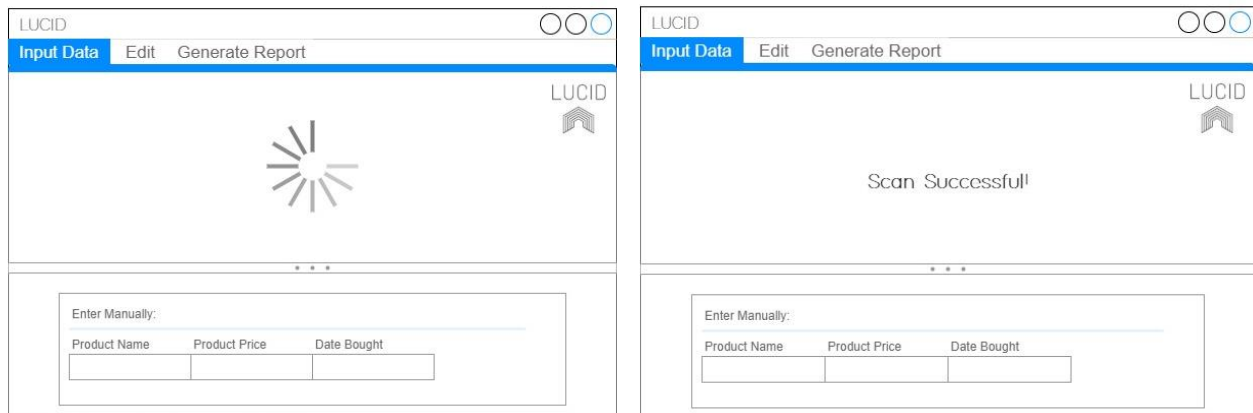
//perform a binary search for the following items in each text file

def SUB binaryTextSearch(f, sItem)
    n = FOR line in f //number of lines in file
    n /= 2
    Found = False
    While Found = False
        FOR each line in f[0:n]
            IF searchItem in line
                Found = True
                return searchItem
                f = f[0:n]
                binaryTextSearch(f, sItem)
            ELSE Found = False
        FOR each line in f[n:]
            IF searchItem in line
                Found = True
```

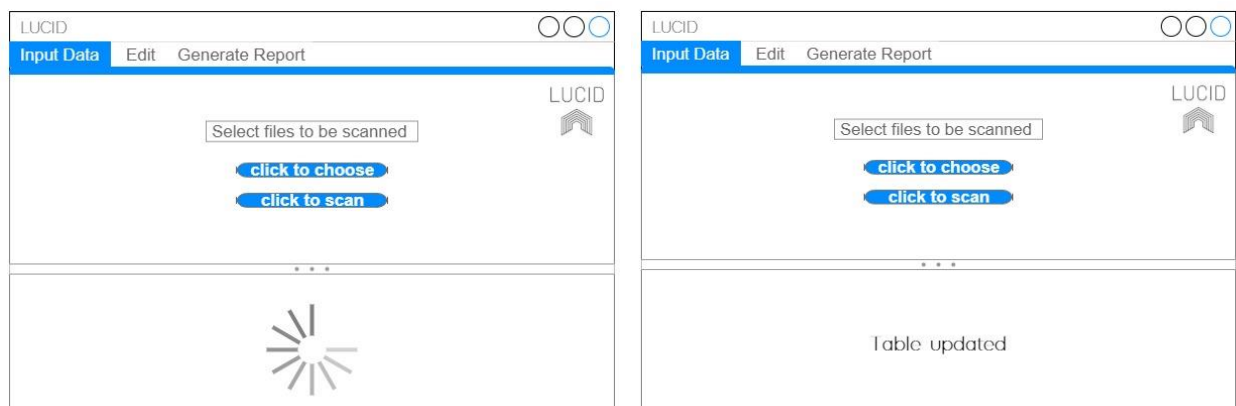
---

## Costs

```
def SUB inputData()  
    choice = input("Would you like to input sales or costs?")  
    IF choice == "costs" THEN  
        SUB inputCosts()  
    ELSEIF choice == "sales" THEN  
        SUB inputSales()
```



Whilst the OCR system scans each receipt, take them off the queue, formats the file, write to the database, a loading image/progress bar will be shown to the user. Once all of this has been complete then a message will be given to the user telling them it has been complete. The user may choose to do anything else now.



If the user has an input for the input fields, they can enter all the values. If not all the values are filled out, then the program will give an error box. Instead of the scanning, the data will be simply placed into the database. This, as expected will be a faster method of storing the data on the database in short term e.g. if you have one item. If you have several items, scanning would be ideal.

## Edit Data



The screenshot shows the LUCID application window with the 'Edit' tab selected. The table contains the following data:

Item	Price	Date
Banana	£1.00	12/03/19
Apples	£2.50	12/03/19
KFC	£3.59	11/03/19
ASDA	£43.20	09/02/19
Water Bill	£129.34	09/02/19
Gas Bull	£34069	09/02/19
Electricity	£240.33	09/02/19
Rent	£700	09/02/19

If the user chooses to edit the data that is stored, they can do so. They will be represented by a table which will be read from the database that it is stored in. The user will be able to scroll throughout the dates that they may need. They can also filter the dates through what they want to see.



The screenshot shows the LUCID application window with the 'Edit' tab selected. The table contains the following data:

Item	Price	Date
Banana	£1.00	12/03/19
Apples	£2.50	12/03/19
KFC	£3.59	11/03/19
ASDA	£43.20	09/02/19
Water Bill	£129.34	09/02/19
Gas Bill	£340.69	09/02/19
Electricity	£240.33	09/02/19
Rent	£700	09/02/19

As we can see from the first image, some of the data extraction may not always be perfect. This data can easily be updated using manually. Once all the updates are made, the values which are updated will change in the database.

```
def SUB editData()
    choice = input()
    //will be a button for them to choose dates
    IF choice == editInput THEN
        SQL(SELECT (date_purchased, total price, location)
            FROM Receipts)
        Display table
        Edit = input()
        IF Edit:
            newDate = input()
            newPrice = input()
            newLocation = input
            unixDate = unix(newDate)
            SQL(UPDATE Receipts WHERE (date_purchased =
newDate, total_price = newPrice, location = newLocation,
unix_date = unixDate)//updates values
            ELSE
                NULL
        ELSEIF choice == editSales THEN
            SQL(SELECT (date_sales, selling_price, item_sold)
                FROM Sales)
            Display table
            Edit = input()
            IF values change
            IF Edit:
                newDate = input()
                newSelling = input()
                newItemSold = input
                unixDate = unix(newDate)
                SQL(UPDATE SALES WHERE (date_sales = newDate,
selling_price = newSelling, location = newLocation, unix_date
= unixDate)
            ELSE
                NULL
```

## Generate Reports

The first screenshot shows the LUCID application interface with the 'Generate Report' tab selected. It prompts the user to 'Please select a time frame:' and displays two calendar grids. The left grid is for October 2018, and the right grid is for January 2020. In the October 2018 grid, the 24th is highlighted in red. In the January 2020 grid, the 19th is highlighted in red. A blue 'Generate Report' button is at the bottom.

The second screenshot shows the same interface, but the main content area is replaced by a large loading spinner, indicating that the report is being processed.

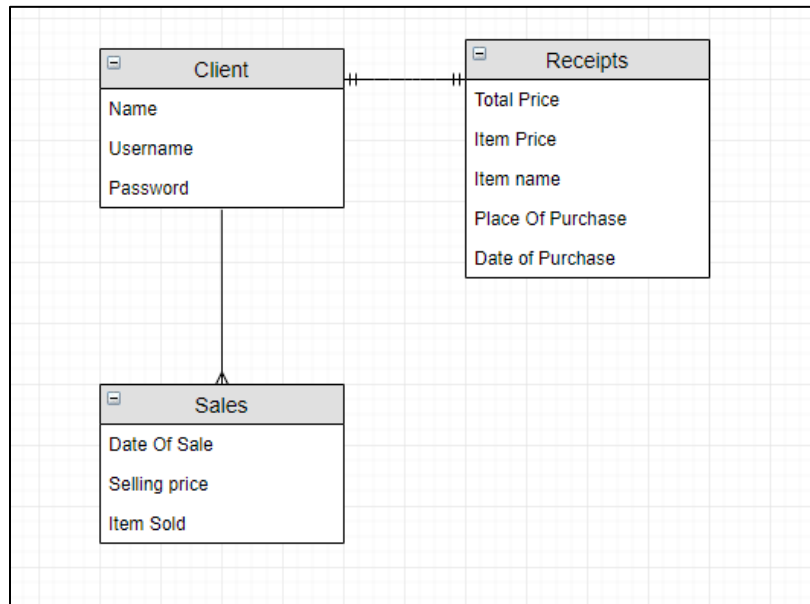
The third screenshot shows the same interface, but the main content area is replaced by a blue button that says 'click to download report'.

The user may choose to generate reports. This is one of the essential parts of the program as my client will need to hand these to HM Customs. There is a typical template of how these letters are handed. A section for all revenue, a section for total costs and a section for profits made. There is a template online where you can simply enter all this information yourself. The program will be doing this for us here. The client can indicate the time frame that should be given to the user. This would process the dates of the sales and costs that have been tracked. A few calculations will be complete, and the final output will be the report. This report will be available to download, and the client can choose to either email the report or print it out – which ever seems the better option. This will then be placed externally and encrypted, where the user can only access old reports on the application.

```
def SUB generateReport()  
    selectDate = False  
    //If they click the button, selectDate becomes true  
    IF selectDate == True THEN  
        dateBegin = input("Beginning date of report: ")  
        dateEnd = input("End date of report: ")  
        SQL(SELECT date FROM Scanned Receipt  
            WHERE date = dateBegin TO date = dateEnd  
            SELECT totalPrice FROM Scanned Receipt  
            WHERE date = dateBegin TO date = dateEnd  
            reportPrice += totalPrice  
            SELECT revenue FROM Sales WHERE  
            date = dateBegin TO date = dateEnd)  
            reportSales += revenue)  
  
        f.open("reportTemplate.doc", "a")  
        f.write(reportPrice where f.readline() == ("Costs:"))  
        f.write(reportSales where f.readline() ==("Sales Made:"))  
        profit = reportSales - reportPrice  
        f.write(profit where f.readline() == ("Profits:"))  
        f.close(as ("Report"+ dateBegin "to" + dateEnd).docx
```

## Data handling

### Entity Attribute Model



The interaction we can see here are between my client and the attributes of the different entities needed.

My Client will have a username and password. This is to ensure that the system they use is secure and only they will be able to access it. Their name will play a role in the log in system, so they have a high level of security which avoids any legal issues such as hackers trying to intercept my client's

data.

Receipts have many attributes as listed. These are a few common attributes which the many different types of receipts have. When an item is purchased by my client, he will receive a receipt, which could have all/most of these details. This is also where the OCR'd data may stay. This will be the data which is read off the receipt.

Sales Made will depend solely on the client. No other external factors will affect his sales, only he can make the changes to this. The attributes listed will help create the reports that he will need by calculating all his profits or see trends between his products to see which is most profitable.

Total Price may be needed if the OCR extraction does not happen properly. A calculation could be made to show how much has been spent, and what the different price of the items are. The gap in the price could be added to the database with the item bought having a name of NULL. This name could then be modified by my client when he checks to see if the extraction was a success.



### First Normal Form

```
tblClient(clientName, username, password)

tblSales(dateOfSale, Revenue, SoldItem)

tblReceipts(datePurchased, itemsPurchased, itemsPrice, totalPrice,
address, contactInfo, paymentMethod, itemPurchased, item Price)
```

The tables in the database are as seen in the ERD. These are the fields of data that will be needed in the programme. They are having unique fields and have no overlap between one another, meaning that they are all unique.

### Second Normal Form

```
tblClient(clientID, clientName, username, password)

tblSales(salesID, dateOfSale, Revenue, SoldItem)

tblReceipts(receiptsID, datePurchased, itemsPurchased, itemsPrice,
totalPrice, address, paymentMethod, itemPurchased, item Price)
```

Here, the first column in all the tables are primary keys. They are not linked and have their own partial dependencies. Now we can separate the data into the necessary tables.

### Third Normal Form

```
tblClient(clientID, clientName, username, password)

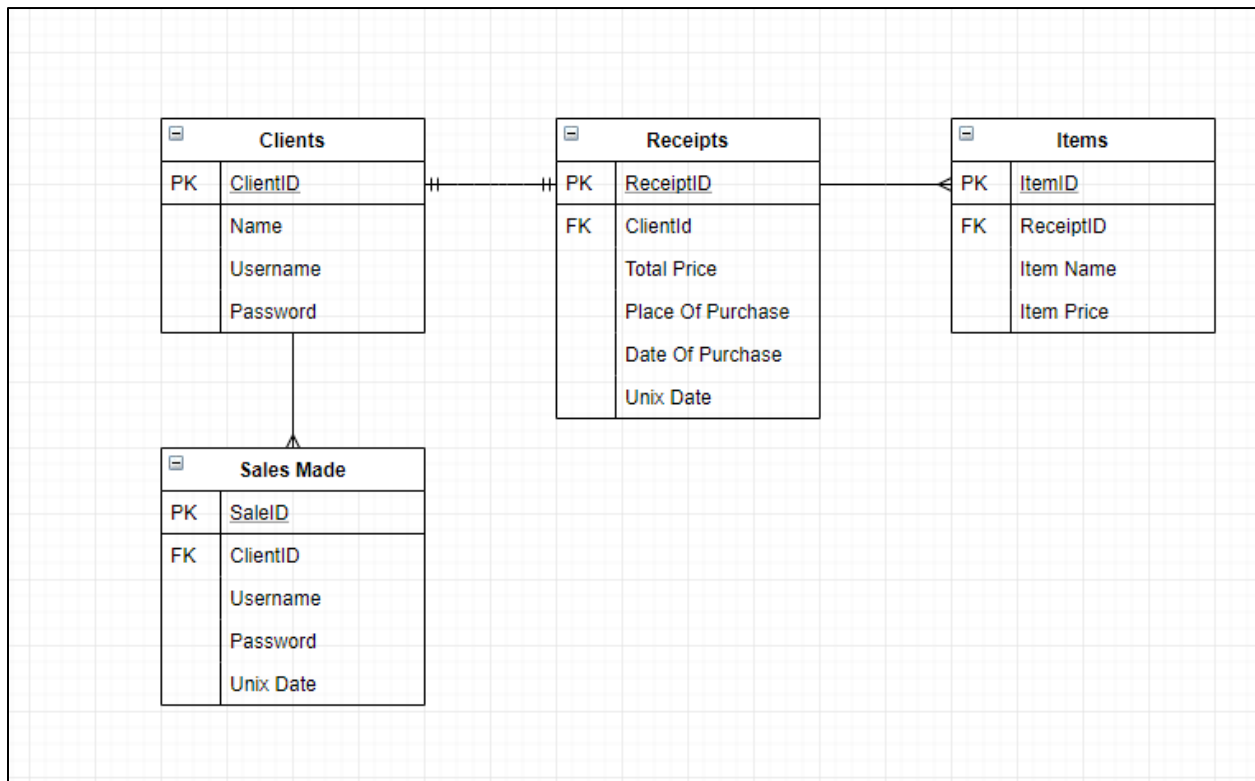
tblSales(saleID, dateOfSale, Revenue, SoldItem, clientID)

tblReceipts(receiptID, datePurchased, totalPrice, address,
paymentMethod, itemPurchased, clientID)

tblItem(itemID, itemPurchased, itemPrice, receiptID)
```

Here, all of the tables each have a primary key, and a foreign key showing what table connects to what. We can clearly see that the both receipts and sales tables connect to the client table, and the item table connects directly to the receipts table. They have no transitive functional dependencies so it is now in third normal form.

## Post-Normalisation Entity Relationship Diagram



The entity 'Item' was needed to be separate from the rest of the entities, and I have done this through giving IDs to both Items and Receipts. This is so we can create a relationship between the item and the receipt of where the item belongs from. Each receipt can also have many items, so I thought it necessary to have a separate entity which can relate back to the receipt table.

Now all the entities have a relationship between them all. Receipts and Items are all initially linked to the client, but "Sales Made" is not. This is because they have no inherent relationship. The only link they have is through the client.

All entities now have the correct ids and are linked to the client. Theoretically, it would be possible to return all the data in the database based on just the clientID e.g. if I wanted to return all of the values on client 1.

## Data Dictionaries for tables

Table Name	tblClient		
Primary Key	ClientID		
Foreign Keys	NA		
Data Item	Data Type	Validation	Example Data
clientID	Integer	none	1,2,3
Client Name	string	NotNull	Zubaid_Amad
username	string	notNull	zAmadxarif12
password	string	notNull	B4n4naKe7chu9\$

For my scenario, there is only one user. However, if there were multiple users using the same program, the client information will be stored on this specific table on the database. This will play into a log in system which will allow the user to keep their specific receipt data secure. When an input is made when the user logs in, the program should search the table for their username and password. If found, it would return to the rest of the program, and if not, it would ask the user again for their input. The client ID creates a relationship between the user's receipts so that they could not get muddled up with another's data. It also has no validation as the client ID will be autoincremented according to the number of clients already inside of the database.

Table Name	tblReceipts		
Primary Key	ReceiptID		
Foreign Keys	ClientID		
Data Item	Data Type	Validation	Example Data
ReceiptID	Integer	Auto Increment	1,2,3
datePurchased	date	In format dd/mm/yy	12/05/2015
unixDateReceipt	Real	Made from date string	121314141.0
TotalPrice	float	To 2 decimal places	54.75

<b>contactInfo</b>	string	None	02031233214
<b>paymentMethod</b>	string	None	Card Payment / Cash
<b>clientID</b>	integer	Is inside table Clients	1,2,3

This table in the database, will store all the receipts data which will be scanned using the OCR engine. Given that all the data has been scanned correctly, and is formatted correctly (see algorithms), in the way that can be written to the database, it will be written. These data items are examples of what is most likely to be on receipts due to the many forms they come in.

ReceiptID will be generated randomly, and will be assigned given the client ID. As this will be generated using the client ID, the validation will create the client ID again. There is no validation for both contactInfo and paymentMethod as they may not be on the receipts. Contact Information could also be an email address, so the validation may not be necessary.

Table Name	tblItem		
<b>Primary Key</b>	itemID		
<b>Foreign Keys</b>	receiptID		
Data Item	Data Type	Validation	Example Data
<b>receiptID</b>	Integer	Is In receipt ID	1,2,3,
<b>itemID</b>	Integer	Autoincrement	1,2,3
<b>Item Name</b>	String	Not Null	Bananas
<b>Item Price</b>	Real	Not Null	3.40

On receipts, we can get items which are singularly bought. The user may choose to place these in the database if he is doing so manually. So, a receipt may have no items or many items. Therefore, it is important to link it to the correct receipt id – this is so we know which receipt the items belong to.

Every time an item is added onto the database, the itemID will also be incremented. This is because as the primary key it must be unique and cannot be the same as any other value in the database.

Each item has its own name and price. This may be useful for graph generation to see how much of what product the user is buying.

Table Name	tblSales		
Primary Key	saleID		
Foreign Keys	clientID		
Data Item	Data Type	Validation	Example
DateOfSale	str	In the format dd/mm/yy	22/09/17
UnixDateSales	real	Made from DateOfSale	12767687980.0
Revenue	str	NOTNULL	"£3.50"
Sold Item	str	NOTNULL	"Pink Scarf" , "Yellow Hat"

The table Sales will be keeping track of all the sales that my client makes. He will need to input these manually into a GUI input box, where three records must be filled out. Not even one of these can be kept empty as they are all important for the reports that must be generated. The date is one of the more important records as this can be cross referenced with the tblScannedReceipts where we can see how much profits were made between the dates given. To perform such queries, the date must be converted into a more convertible data type. The Unix Date is the date that will be used to plot the graphs as we will actually be able to order the graphs based on the real values

## SQL Statements

```
CREATE TABLE Client (
  clientName varchar(30),
  username varchar(255)NOTNULL,
  password varchar(255)NOTNULL,
  clientID int NOTNULL PRIMARY KEY
)
```

```
CREATE TABLE Items(
  itemID int PRIMARY KEY,
  itemName str varchar(30),
  itemPrice REAL,
  receiptID int,
  receiptID REFEEERENCES (RECEIPTS)
  FOREIGN KEY)
```

Here I have created a table in the database, acting as the first entity in the EAM diagram shown.

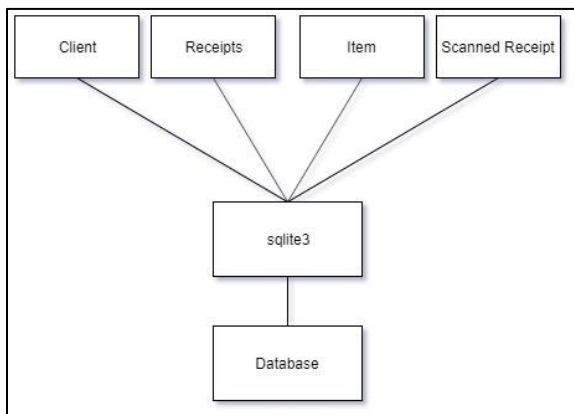
The receipt ID will be created to form a relationship between the two tables. This value could be generated randomly or use a key part of the receipt to do so. The receipt ID column will be taken to create the receipts table. This will make it easier to look up receipts when my client requests to see them. The rest of the columns are added in manually to create the table.

```
CREATE TABLE
```

```
Receipts(
  int PRIMARY KEY,
  int FOREIGN KEY
  Client(client_id),
  sed str,
  rchase str,
  REAL
```

```
CREATE TABLE SALES (
  sales_id int PRIMARY KEY,
  client_id int FOREIGN KEY
  REFERENCES Client(client_id),
  date_sales REAL,
  item_sold text)
```

```
CREATE TABLE Sales(
  ADD dateOfSale date NOTNULL,
  ADD Revenue varchar NOTNULL,
  ADD SoldItem varchar NOTNULL,
  );
SELECT clientID INTO Sales FROM
Client
```

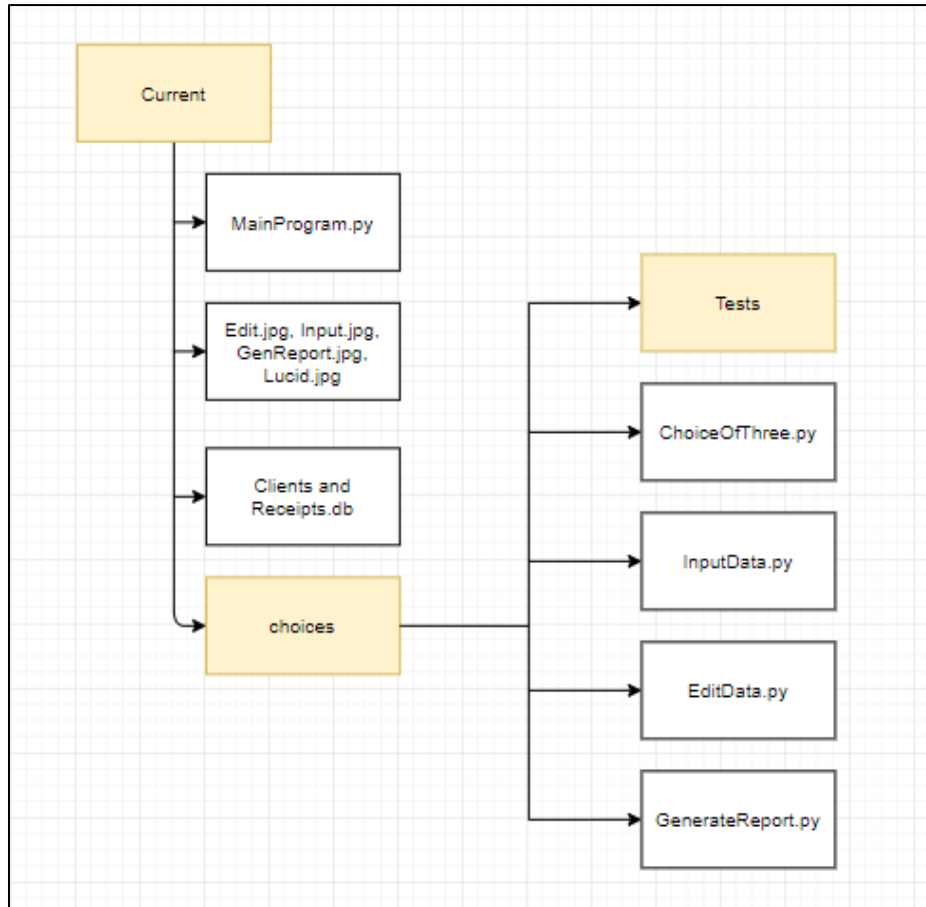


There are the different tables which are created and then stored in the databases. All of which have a connection with each other. This may be through a foreign or primary key. The DBMS is

sqlite3 which will allow me to interact with the different tables in the database. It will also play a role when we create the reports as the data will need to flow to and from the databases.

## Technical Solution


### Folder Management



I have chosen to lay out my folders as shown above. The yellow coloured boxes are the folders, and the arrows show the contents of the folders.

In folder 'Current' the initial file is stored, where the program begins. This calls in image 'Lucid.jpg' to present the logo on top of the log in page. When the log in system is presented to the user and the fields are submitted, the program will go and check if the inputs are inside of the database – which is also stored inside of the database. If the details are correct, the program proceeds into the next folder.

The 'choices' folder contains the main python files which run the program. If the details are correct, 'ChoiceOfThree.py' is run where the user is asked to choose which service, they would like to use. In the program, the directory of the program is changed to the 'Current' folder temporarily. This is so they can access both the database and images. All of the files in this folder

 have the three options in the corner of the program, and they will be able to access it without problem.



## Database Management

```
1  #database setup
2
3  import sqlite3 as sq
4  dbFin = sq.connect('Clients and Receipts.db')
5  c = dbFin.cursor()
6
7  # Create Table - Client Info, using client_id
8  c.execute('''CREATE TABLE IF NOT EXISTS CLIENTS
9              ([client_id] INTEGER PRIMARY KEY AUTOINCREMENT,
10              [Client_Name] text,
11              [Username] text,
12              [Password] text)''')
13
14  #Create Table for sales made using foreign key client_id
15
16  c.execute('''CREATE TABLE IF NOT EXISTS SALES_MADE
17              ([client_id] INTEGER,
18              [Sale_ID] INTEGER PRIMARY KEY AUTOINCREMENT,
19              [Date_Of_Sale] text,
20              [unix_date_sales] REAL,
21              [Revenue] REAL,
22              [ItemSold] text,
23              CONSTRAINT fk_client
24              FOREIGN KEY (client_id) REFERENCES CLIENTS(client_id))''')
25
26  # create Table Receipts using client_id as a foreign key
27  # use receipt_id as a primary key and autoincrement
28
29  c.execute('''CREATE TABLE IF NOT EXISTS RECEIPTS
30              ([client_id] INTEGER,
31              [receipt_id] INTEGER PRIMARY KEY AUTOINCREMENT,
32              [total_price] REAL,
33              [place_of_purchase] text,
34              [Date_of_Purchase] text,
35              [unix_date_purchase] REAL,
36              CONSTRAINT fk_client
37              FOREIGN KEY(client_id) REFERENCES CLIENTS(client_id))''')
38
39  #create table for item use item_id as a primary key
40  #use receipt_id as a foreign key to set up a link
41
42  c.execute('''CREATE TABLE IF NOT EXISTS ITEM(
43              [receipt_id] INTEGER,
44              [item_id] INTEGER PRIMARY KEY AUTOINCREMENT,
45              [item_purchased] text,
46              [item_price] REAL,
47              CONSTRAINT fk_client
48              FOREIGN KEY(receipt_id) REFERENCES RECEIPTS(receipt_id))''')
49
50  #below table will be the one where the user will be able to see
51
52
53  #when adding the data just take the columns that exist and place it into the other tables
54
55  dbFin.commit()
56
```

Each table has been made beginning with the client. This is because the client is the entity which gives its primary key as a foreign key but does not use a foreign key itself. This means I was able to create the Receipts and Sales\_Made tables. These tables both use the client\_id as a foreign key, whilst having their own primary keys; receipt\_id and sales\_id respectively. This means that

we were able to create the final table for Items using receipt\_id as a foreign key. This means that all my tables were created.

I had set constraints on all the tables which have a foreign key. This is to ensure that either a receipt or sale would have a client that is in the database. For example, a client (4) that does not exist in the clients table, will not be able to insert values into the database. This works similarly for the receipt\_id constraint on the Items Table – an item with a receipt id that does not exist in the receipt table cannot be added.

## Initial Program Run – Log in Page – MainProgram.py

```
1. from tkinter import *
2. import tkinter as tk
3. from PIL import ImageTk, Image
4. from tkinter.font import Font
5. from tkinter import messagebox
6. import os
7. import sqlite3 as sq
8.
9. #passes on client_id to rest of program
10.     global client_id
11.
12.
13.     class Base:
14.         def __init__(self, master):
15.             master.title("Lucid")
16.             master.configure(background = '#ffffff')
17.             master.config(height = 1600, width = 800)
18.             master.state('zoomed')
19.             #initiliasse tkinter variables
20.             username = StringVar()
21.             password = StringVar()
22.
23.             #username input box
24.             Label(text="Username", border = 0, font = "Adam",
25. bg = "white").place(relx=0.45, rely=0.5, anchor='center')
26.             usernameIn = tk.Entry(root, textvariable =
27. username)
28.             usernameIn.place(relx = 0.5, rely = 0.55, anchor =
29. 'center')
30.
31.             #password input box
```

```
29.         Label(text="Password", border = 0, font = "Adam",
30.             bg = "white").place(relx=0.45, rely=0.6, anchor='center')
31.         passwordIn = tk.Entry(root, textvariable =
32.             password, show="*")
33.         passwordIn.place(relx = 0.5, rely = 0.65, anchor =
34.             'center')
35.         #submit button
36.         submitDetails = Button(root, text = "Submit
37.             details")
38.         submit = Button(root, text="Submit", font = "Adam"
39.             , command= lambda: self.checkDetails(username.get(),
40.             password.get()))
41.         submit.place(relx = 0.5, rely = 0.7, anchor =
42.             'center')
43.         def checkDetails(self, username, password):
44.             db = sq.connect('Clients and Receipts.db')
45.             c = db.cursor()
46.             user = ("SELECT * FROM CLIENTS WHERE Username = ?
47.             AND Password = ? ")
48.             #finds the username and password input the client
49.             has placed in
50.             c.execute(user, [username, password])
51.             check = c.fetchall()
52.             if check:
53.                 #if it is found, access is allowed,
54.                 ChoiceOfThree.py is opened
55.                 messagebox.showinfo("Access Granted", "Click
56.                 'OK' to continue")
57.                 item = []
58.                 for row in check:
59.                     client_id = row[0]
60.                     root.destroy()
61.                     import choices.ChoiceOfThree
62.             else:
63.                 messagebox.showerror("Error", "Incorrect
64.                 Credentials")
65.                 #if username is not found^
66.
67.         root = Tk()
```

```
60.      #Object instantiated over base class
61.      LogInPage = Base(root)
62.
63.      #logo
64.      logo = PhotoImage(file='Lucid[final1].png')
65.      Label(image=logo, border = 0).place(relx=0.5, rely=0,
        anchor="n")
66.
67.      root.mainloop()
```

## ChoiceOfThree.py

```
1. from tkinter import *
2. import tkinter as tk
3. from PIL import ImageTk, Image
4. from tkinter import ttk
5. import sys
6. import os
7.
8.
9. os.chdir('C:/Users/zubi_/Desktop/[project_skyfall]/Programming/GU
  I/Current')
10.
11.     root = Tk()
12.     root.title("Lucid")
13.     root.configure(background = '#ffffff')
14.     root.config(height = 1600, width = 800)
15.     root.state('zoomed')
16.
17.     #if the user chooses Input/Edit/Generate Report, it will be
        loaded
18.     #current GUI will be closed
19.
20.     def InputData():
21.         root.destroy()
22.         import InputData
23.
24.     def EditData():
25.         root.destroy()
26.         import EditData
27.
```

```
28.     def GenerateReport():
29.         root.destroy()
30.         import GenerateReport
31.
32.         inputImage = ImageTk.PhotoImage(Image.open("input.png"))
33.         editImage = ImageTk.PhotoImage(Image.open("edit.png"))
34.         genImage = ImageTk.PhotoImage(Image.open("genreport.png"))
35.         choice1 = Button(root, image = inputImage,
36.             command=InputData).place(relx = 0.04, rely = 0, anchor= "n")
37.         choice2 = Button(root, image = editImage,
38.             command=EditData).place(relx = 0.115, rely = 0, anchor = "n")
39.         choice3 = Button(root, image = genImage, command =
40.             GenerateReport).place(relx = 0.19, rely = 0, anchor = "n")
41.         text = Label(text="Please choose from the following
42.             options:", border = 0, font = ("Adam", 40), bg =
43.             "white").place(relx=0.5, rely=0.5, anchor='center')
44.
45.         root.mainloop()
```

## InputData.py

```
1. from tkinter import *
2. import tkinter as tk
3. from PIL import ImageTk, Image
4. from tkinter import ttk
5. from tkinter import filedialog
6. from tkinter import messagebox
7. import sys
8. import pytesseract
9. from pytesseract import Output
10.     import re
11.     import cv2
12.     import matplotlib
13.     import os
14.     import sqlite3 as sq
15.     import time
16.     import datetime
17.
18.
19.     ##hardcoded
20.     client_id = 1
21.
22.
```

```
23.     root = Tk()
24.     root.title("Lucid - Input Data")
25.     root.configure(background = '#ffffff')
26.     root.config(height = 1600, width = 800)
27.     root.state('zoomed')
28.
29.     os.chdir('C:/Users/zubi_/Desktop/[project_skyfall]/Programm
    ing/GUI/Current')
30.
31.     global filenames
32.     fileQueue = []
33.     #is empty each time data is input
34.
35.     #if the user chooses to use other processes, the current
    gui will close,
36.     #chosen gui will load
37.     def InputData():
38.         root.destroy()
39.         import InputData
40.
41.     def EditData():
42.         root.destroy()
43.         import EditData
44.
45.     def GenerateReport():
46.         root.destroy()
47.         import GenerateReport
48.
49.
50.     inputImage = ImageTk.PhotoImage(Image.open("input.png"))
51.     editImage = ImageTk.PhotoImage(Image.open("edit.png"))
52.     genImage = ImageTk.PhotoImage(Image.open("genreport.png"))
53.
54.
55.     #input button disabled as it is currently being used
56.     inputData = Button(image = inputImage, command=InputData,
    state = DISABLED).place(relx = 0.04, rely = 0, anchor= "n")
57.     EditData = Button(image = editImage,
    command=EditData).place(relx = 0.115, rely = 0, anchor = "n")
58.     GenReport = Button(image = genImage, command =
    GenerateReport).place(relx = 0.19, rely = 0, anchor = "n")
59.
60.     #place to see which files are ready to scan
```

```
61.     fileView = Listbox(root, width = 40, height = 20, font =  
        "Adam")  
62.     fileView.place(relx = 0.1, rely = 0.3)  
63.  
64.  
65.     #function to allow user to choose the images they want to  
        scan  
66.     #this will place these in the Listbox  
67.     def fileroot():  
68.         root.filename =  
        filedialog.askopenfilenames(initialdir="/", filetypes=(("png  
        files", "*.png"), ("jpg files", "*.jpg")))  
69.         files = list(root.filename)  
70.         for i in range(len(files)):  
71.             fileView.insert(i, files[i])  
72.             fileQueue.append(files[i])  
73.  
74.     #if tesseract is used, the following will occur  
75.     def storeToTes(date, subtotal, location, item, itemPrice):  
76.         db = sq.connect('Clients and Receipts.db')  
77.         #connection established  
78.         c = db.cursor()  
79.  
80.         try: #if the date can be converted in unix format,  
        program will place the following in the database  
81.             unixDate =  
            time.mktime(datetime.datetime.strptime(date,  
            "%d/%m/%y").timetuple())  
82.             query = ''' INSERT INTO RECEIPTS  
83.                 (client_id ,Date_of_Purchase,  
            unix_date_purchase, total_price, place_of_purchase)  
84.                 VALUES  
85.                 (?, ?, ?, ?, ?)  
86.                 '''  
87.             receiptDetails = (client_id, str(date),  
            float(unixDate), str(subtotal), str(location))  
88.             except: #if the date cannot be found, unix cannot be  
            calculated hence;  
89.             query = ''' INSERT INTO RECEIPTS  
90.                 (client_id, Date_of_Purchase,  
            total_price, place_of_purchase)  
91.                 VALUES  
92.                 (?, ?, ?, ?)
```

```
93.                                     '''
94.             receiptDetails = (client_id, str(date),
    str(subtotal), str(location))
95.
96.
97.             c.execute(query, receiptDetails)
98.
99.             #now that receipt has been inserted, we can use its
    receipt_id
100.            c.execute('SELECT MAX(receipt_id) FROM RECEIPTS')
101.            temp = []
102.            for row in c.fetchall():
103.                temp.append(int(row[0]))
104.                receipt_id = int(temp[0])
105.
106.
107.            #for the number of items found, place them into the
    items table
108.            #using the receipt_id
109.            for i in range(len(itemPrice)):
110.                query = (''INSERT INTO ITEM
111.                    (receipt_id, item_purchased,
112.                        item_price)
113.                        VALUES
114.                        (?, ?, ?)'' )
114.                items = (receipt_id, item[i], itemPrice[i])
115.                c.execute(query, items)
116.
117.            db.commit()
118.            db.close()
119.
120.            messagebox.showinfo("Success", "Scanning has been
    successful")
121.
122.
123.            #if the user were to choose manual input for receipts
124.            def storeToReceipts(date, subtotal, location):
125.                db = sq.connect('Clients and Receipts.db')
126.                c = db.cursor()
127.
128.                try: #exception handler to see if a unix date can
    be created
```



```
129.         unixDate =
130.         time.mktime(datetime.datetime.strptime(date,
131.         "%d/%m/%y").timetuple())
132.         print(unixDate)
133.         query = ''' INSERT INTO RECEIPTS
134.         (client_id ,Date_of_Purchase,
135.         unix_date_purchase, total_price, place_of_purchase)
136.         VALUES
137.         (?,?,?,?,?)'''
138.         receiptDetails = (client_id, str(date),
139.         float(unixDate), str(subtotal), str(location))
140.         except:
141.             #only way the unix date is calculated is using
142.             the date in format dd/mm/yy
143.             messagebox.showerror("Error","Please enter the
144.             date in format dd/mm/yy")
145.
146.         c.execute(query, receiptDetails)
147.         db.commit()
148.         db.close()
149.         messagebox.showinfo("Success", "Scanning has been
150.         successful")
151.
152.         #when the images are clicked for scanning the subroutine
153.         function is run
154.         def tesseractOCR ():
155.             if len(fileQueue) > 0:
156.                 for i in range(len(fileQueue)):
157.                     path = fileQueue[i]
158.                     image = cv2.imread(path)
159.                     grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
160.                     #image converted into black and white for a
161.                     higher accuracy
162.                     grey = cv2.threshold(grey, 0, 255,
163.                     cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
164.
165.
166.                     if path[-4:] == ".jpg":
167.                         filename = "{}.jpg".format(os.getpid())
168.                         #checks if it is in an image format
```

```
161.         elif path[-4:] == ".png":
162.             filename = "{}.png".format(os.getpid())
163.
164.             cv2.imwrite(filename, grey)
165.
166.
167.             #path specifying where tesseract is
168.             pytesseract.pytesseract.tesseract_cmd =
169.                 r'C:\Program Files (x86)\Tesseract-OCR\tesseract.exe'
170.
171.             #stores OCR'd content into a dictionary
172.             d = pytesseract.image_to_data(filename,
173.                 output_type=Output.DICT)
174.
175.             os.remove(filename)
176.             keys = list(d.keys())
177.
178.             #following are regular expressions for the
179.             date, prices, names
180.             datePattern = '^([0-9]|12)[0-9]|3[01])/([0-9]|1[012])/([12][0-9])\d\d$'
181.             pricePattern = '[+]?[0-9]+\.[0-9]+'
182.             doorNum = '[0-9]{1,3}(?![0-9])'
183.             Names = '[A-Z]+[a-z]+$'
184.
185.             #subroutine to look for substring inside an
186.             item inside a list
187.             def findSubstring(List, string):
188.                 return string in List
189.
190.             date = ''
191.             prices = []
192.             numbers = []
193.             names = []
194.             checkPrice = 0
195.             values = d['text']
196.             # gets rid of empty spaces in the list from
197.             dictionary
198.             values = list(filter(None, values))
199.             print(values)
200.
201.             #looks for the total price index by looking for
202.             index of following words
```

```
197.         if len(values) != 0:
198.             try:
199.                 priceIndex = (values.index("Subtotal"))
200.             except:
201.                 priceIndex = 0
202.
203.             try:
204.                 priceIndex = (values.index("Total")) +
205.                 1
206.             except:
207.                 priceIndex = 0
208.
209.         totalPrice = values[priceIndex]
210.
211.
212.         #looks for index of location
213.         for i in range(len(values)):
214.             if findSubstring(values[i], 'Road') == True
215.             or findSubstring(values[i], 'road') == True or
216.             findSubstring(values[i], 'Rd') == True:
217.                 placeIndex = i
218.
219.         for i in range(len(values)):
220.             if re.match(datePattern, values[i]):
221.                 date = values[i]
222.             # any date on the receipt
223.             if re.match(pricePattern, values[i]):
224.                 prices.append(values[i])
225.                 #list of prices to place in db
226.             if re.match(doorNum, values[i]):
227.                 numbers.append(values[i])
228.                 #list of all the numbers in receipt
229.             if re.match(Names, values[i]):
230.                 names.append(values[i])
231.                 #list of all words with capital letters
232.         in receipt
233.         checkList = values[:priceIndex]
234.         # everything before the word 'total' or eqv.
```

```
234.          #following uses regex to check if there is a
float.
235.          #returns its index and places in a new list
236.          #index of 1 before that is the name of the
item, also placed in list
237.          items = []
238.          itemsPrices = []
239.          for i in range(len(checkList)):
240.              if re.match(pricePattern, checkList[i]):
                #if there is a float
241.                  items.append(checkList[i-1])
242.                  itemsPrices.append(checkList[i])
243.
244.
245.
246.              if int(values[placeIndex - 2]) ==
int(values[placeIndex - 2]):
247.                  address = values[placeIndex - 2] ,
values[placeIndex - 1] , values[placeIndex]
248.              elif int(values[placeIndex + 1]):
249.                  address = values[placeIndex + 1]
250.              else:
251.                  None
252.
253.                  placeOfPurchase = names[0]
254.                  address = str(placeOfPurchase) + str(address)
255.                  storeToTes(date, totalPrice, address, items,
itemsPrices)
256.                  fileView.delete(0, END)
257.                  print(date)
258.                  # clears the displayed box of any receipts
259.              else:
260.                  messagebox.showerror("Error", "No files have been
selected")
261.                  #if there is nothing in the listbox, an error box
is displayed
262.
263.          def storeToSales(gui, Date, Price, Item):
264.              db = sq.connect('Clients and Receipts.db')
265.              c = db.cursor()
266.
267.              try: #exception handling to see if a date exists
```

```
268.         unixDate =
time.mktime(datetime.datetime.strptime(Date,
"%d/%m/%y").timetuple())
269.         query = '''INSERT INTO SALES_MADE
270.         (client_id, Date_Of_Sale, unix_date_sales, Revenue,
ItemSold)
271.         VALUES
272.         (?, ?, ?, ?, ?)
273.         WHERE (place_of_purchase) = 'none' '''
274.         sale = (client_id, str(Date),
float(unixDate), float(Price), str(Item))
275.
276.         c.execute(query, sale)
277.         db.commit()
278.         db.close()
279.
280.         gui.destroy()
281.         messagebox.showinfo("Success", "Sales have been
added")
282.     except:
283.         gui.destroy()
284.         messagebox.showerror("Error", "Please enter the date
in dd/mm/yy format")
285.         #unicode date created to allow sorting of the dates
286.
287.     global receipt_id
288.     receipt_id = []
289.     none = 0
290.
291.
292.     none += 1
293.     return none
294.
295.     #this subroutine runs when there receipts have items
//manual input
296.     def storeToItems(name, price, receipt_id, none):
297.         print(none)
298.         db = sq.connect('Clients and Receipts.db')
299.         c = db.cursor()
300.
301.         query = ('''SELECT max(receipt_id)
302.                 FROM RECEIPTS''')
303.         # returns the largest receipt_id
```

```
304.         c.execute(query)
305.         for row in c.fetchall():
306.             receipt_id.append(int(row[0]) + 1)
307.             print(receipt_id)
308.             receipt_id = int(receipt_id[0])
309.             #this will be the new receipt id
310.
311.             #This creates the receipt_id for items to be placed
312.             c.execute('' INSERT INTO RECEIPTS
313. (place_of_purchase)
314.                               VALUES (?)'', str(None))
315.             query2 = (''INSERT INTO ITEM (receipt_id,
316. item_purchased, item_price)
317.                               VALUES (?, ?, ?)'')
318.             #inserts each item into the database^
319.             values = (receipt_id, name, price)
320.
321.             c.execute(query2, values)
322.
323.             db.commit()
324.             c.close()
325.
326.     def inputItems():
327.         inputItems = Toplevel()
328.         inputItems.configure(background = '#ffffff')
329.
330.         #name of item bought
331.         Label(inputItems, text = 'Item Name', font = 'Adam', bg
332. = 'white').grid(row = 1, column = 1)
333.         itemName = Entry(inputItems)
334.         itemName.grid(row = 1, column = 2)
335.
336.         #price of item bought
337.         Label(inputItems, text = 'Item Price', font = 'Adam',
338. bg = 'white').grid(row = 2, column = 1)
339.         itemPrice = Entry(inputItems)
340.         itemPrice.grid(row = 2, column = 2)
341.
342.         #runs function to store in database
```

```
341.         Submit = Button(inputItems, text = 'Submit', font =
        'Adam', command = lambda: storeToItems(itemName.get(),
        itemPrice.get(), receipt_id, none)).grid(row = 3, column = 2)
342.
343.         inputItems.mainloop()
344.
345.
346.     def inputSales():
347.         inputSale = Toplevel()
348.
349.         #date of item sold
350.         Label(inputSale, text = 'Date', font = 'Adam').grid(row
        = 1, column = 1)
351.         Date = Entry(inputSale)
352.         Date.grid(row = 1, column = 2)
353.
354.         #price of item sold
355.         Label(inputSale, text = 'Selling Price', font =
        'Adam').grid(row = 2, column = 1)
356.         Price = Entry(inputSale)
357.         Price.grid(row = 2, column = 2)
358.
359.         #item sold
360.         Label(inputSale, text = 'Item Sold', font =
        'Adam').grid(row = 3, column = 1)
361.         Item = Entry(inputSale)
362.         Item.grid(row = 3, column = 2)
363.
364.         #button leading to function which stores to database
365.         Button(inputSale, text = 'Submit', font =
        'Adam', command = lambda: storeToSales(inputSale, Date.get(),
        Price.get(), Item.get())).grid(row = 4, column = 2)
366.
367.         inputSale.mainloop()
368.
369.     def inputReceiptData():
370.         #Pop up window to input receipt data
371.         inputRD = Toplevel()
372.         inputRD.configure(background = '#ffffff')
373.         #date of item sold
374.         Label(inputRD, text = 'Date', font = 'Adam', bg =
        'white').grid(row = 1, column = 1)
375.         Date = Entry(inputRD)
```

```
376.         Date.grid(row = 1, column = 2)
377.
378.         #price of item sold
379.         Label(inputRD, text = 'Subtotal', font = 'Adam', bg =
          'white').grid(row = 2, column = 1)
380.         Subtotal = Entry(inputRD)
381.         Subtotal.grid(row = 2, column = 2)
382.
383.         #item sold
384.         Label(inputRD, text = 'Address', font = 'Adam', bg =
          'white').grid(row = 3, column = 1)
385.         Address = Entry(inputRD)
386.         Address.grid(row = 3, column = 2)
387.
388.
389.         Item = Button(inputRD, text = 'Items On Receipt', font
          = 'Adam', command = lambda: inputItems()).grid(row = 4, column =
          2) #, command = lambda: inputItems())
390.         Submit = Button(inputRD, text = 'Submit', font =
          'Adam', command = lambda: storeToReceipts(Date.get(),
          Subtotal.get(), Address.get())).grid(row = 4, column = 1)
391.
392.
393.         inputRD.mainloop()
394.
395.
396.
397.         fileChoose = Button(text = "Choose files", font = 'Adam',
          command = fileroot).place (relx = 0.1, rely = 0.8)
398.         runOCR = Button(text = "Run Scanner", font = 'Adam',
          command = tesseractOCR).place (relx = 0.25, rely = 0.8)
399.         input_sales = Button(text = "Input Sales", font = ('Adam',
          20), command = inputSales).place(relx = 0.7, rely = 0.2)
400.         input_costs = Button(text = "Input Receipt Data", font =
          ('Adam', 20), command = inputReceiptData).place(relx = 0.7, rely
          = 0.6)
401.         pick = Label(text = "Please select images for scanning",
          border = 0, font = ("Adam", 26), bg = "white").place( relx=0.05,
          rely = 0.2 )
402.
403.         root.mainloop()
```



## Edit Data.py

```
1. from tkinter import *
2. import tkinter as tk
3. from PIL import ImageTk, Image
4. from tkinter import ttk
5. from tkinter import messagebox
6. import sys
7. import sqlite3 as sq
8. import os
9. import datetime
10. import time
11.
12. root = Tk()
13. root.title("Lucid - Edit Data")
14. root.configure(background = '#ffffff')
15. root.config(height = 1600, width = 800)
16. root.state('zoomed')
17.
18. os.chdir('C:/Users/zubi_/Desktop/[project_skyfall]/Programming/GUI/Current')
19.
20.
21. #if user clicks on Input/Edit/GenerateReport then the file will load
22. def InputData():
23.     root.destroy()
24.     import InputData
25.
26. def EditData():
27.     root.destroy()
28.     import EditData
29.
30. def GenerateReport():
31.     root.destroy()
32.     import GenerateReport
33.
34. inputImage = ImageTk.PhotoImage(Image.open("input.png"))
35. editImage = ImageTk.PhotoImage(Image.open("edit.png"))
36. genImage = ImageTk.PhotoImage(Image.open("genreport.png"))
37.
38.
39.
40. root1 = Button(root, image = inputImage, command=InputData)
41. root1.place(relx = 0.04, rely = 0, anchor= "n")
```

```
42.
43.
44. root2 = Button(root, image = editImage, command=EditData, state = DISABLED)
45. root2.place(relx = 0.115, rely = 0, anchor = "n")
46.
47.
48. root3 = Button(root, image = genImage, command = GenerateReport)
49. root3.place(relx = 0.19, rely = 0, anchor = "n")
50.
51.
52. receipts = "Clients and Receipts.db"
53.
54.
55.
56.
57. #Function to execute database queries
58.
59. def runQ(query, para=()):
60.     with sq.connect(receipts) as conn:
61.         cursor = conn.cursor()
62.         result = cursor.execute(query, para)
63.         conn.commit()
64.     return result
65.
66. #function to get table from database and display in treeview
67. def getTable(Table, query):
68.     records = Table.get_children()
69.     print(records)
70.     for elements in records:
71.         Table.delete(records)
72.
73.     tblRows = runQ(query)
74.
75.     try:
76.         #this exception handler checks if it can place the values in the
database
77.         for row in tblRows:
78.             Table.insert('', 'end', values=(row[0], row[1], row[2])) #it works
here for receipts table
79.     except:
80.         for row in tblRows:
81.             Table.insert('', 'end', values=(row[0], row[1])) #works here for
the Items table
```

```
82.
83. def deleteRecord():
84.     try: #if a value is not selected an error is given
85.         sqTable.item(sqTable.selection())['values'][0]
86.     except IndexError as e:
87.         messagebox.showerror("Error", "Select a record to delete")
88.         return
89.     record = sqTable.item(sqTable.selection())['values'][0]
90.     record2 = sqTable.item(sqTable.selection())['values'][1]
91.     #deletes records where the user has selected them
92.     que = 'DELETE FROM RECEIPTS WHERE total_price = ? and place_of_purchase =
    ?'
93.     runQ(que, (record, record2))
94.     #runQ is called and it is stored
95.     messagebox.showinfo("Success", "Record has been successfully deleted")
96.
97.
98.     #all items from the table are removed
99.     sqTable.delete(*sqTable.get_children())
100.
101.     que = 'SELECT total_price, place_of_purchase, Date_of_Purchase FROM
    RECEIPTS'
102.     tblRows = runQ(que)
103.
104.     #updated rows are pulled onto the table
105.     for row in tblRows:
106.         sqTable.insert('', 'end', values=(row[0], row[1], row[2]))
107.
108.
109.
110.     def editReceipt():
111.         try: #if an item is not selected an error is given
112.             sqTable.item(sqTable.selection())['values'][0]
113.         except IndexError as e:
114.             messagebox.showerror("Error", "Select a record to edit")
115.             return
116.
117.         #variables assigned to the selected record
118.         record = sqTable.item(sqTable.selection())['text']
119.         currentPrice = sqTable.item(sqTable.selection())['values'][0]
120.         currentLocation = sqTable.item(sqTable.selection())['values'][1]
121.         currentDate = sqTable.item(sqTable.selection())['values'][2]
122.
```

```
123.
124.         editRecord = Toplevel()
125.
126.         editRecord.title = 'Lucid - Edit Data'
127.         editRecord.configure(background = '#ffffff')
128.
129.         #old total price
130.         Label(editRecord, text = 'Current Total Price:', font =
            'Adam').grid(row = 0, column = 1)
131.         Entry(editRecord, textvariable = StringVar(editRecord, value =
            currentPrice), state = 'readonly').grid(row = 0, column = 2)
132.
133.         #updated total price
134.         Label(editRecord, text = 'New Total Price:', font = 'Adam').grid(row
            = 1, column = 1)
135.         newPrice = Entry(editRecord)
136.         newPrice.grid(row = 1, column = 2)
137.
138.         #current location
139.         Label(editRecord, text = 'Current Location:', font = 'Adam').grid(row
            = 2, column = 1)
140.         Entry(editRecord, textvariable = StringVar(editRecord, value =
            currentLocation), state = 'readonly').grid(row = 2, column = 2)
141.
142.         #updated location
143.         Label(editRecord, text = 'New Location:', font = 'Adam').grid(row =
            3, column = 1)
144.         newLocation= Entry(editRecord)
145.         newLocation.grid(row = 3, column = 2)
146.
147.         #old date
148.         Label(editRecord, text = 'Current Date:', font = 'Adam').grid(row =
            4, column = 1)
149.         Entry(editRecord, textvariable = StringVar(editRecord, value =
            currentDate), state = 'readonly').grid(row = 4, column = 2)
150.
151.         #updated date
152.         Label(editRecord, text = 'New Date:', font = 'Adam').grid(row = 5,
            column = 1)
153.         newDate= Entry(editRecord)
154.         newDate.grid(row = 5, column = 2)
155.
156.         #subroutine to update record is called
```

```
157.         Button(editRecord, text = 'Update', font = 'Adam', command = lambda:
updateRecord(editRecord, currentPrice, currentLocation, currentDate,
newPrice.get(), newLocation.get(), newDate.get())).grid(row = 6, column = 2,
sticky = W)
158.
159.
160.         editRecord.mainloop()
161.
162.
163.
164.         def updateRecord(gui, oldPrice, oldLocation, oldDate, newPrice,
newLocation, newDate):
165.             try:#checks if the date can be converted to unix, if not an error is
displayed
166.                 #following uses old values to update to new values
167.                 unixDateSales = time.mktime(datetime.datetime.strptime(newDate,
"%d/%m/%y").timetuple())
168.                 query = 'UPDATE RECEIPTS SET total_price = ?, place_of_purchase =
?, Date_of_Purchase = ?, unix_date_purchase = ? WHERE total_price = ? and
place_of_purchase = ?'
169.                 parameters = (newPrice, newLocation, newDate, unixDateSales,
oldPrice, oldLocation)
170.                 runQ(query, parameters)
171.
172.                 #every item from the table is deleted
173.                 sqTable.delete(*sqTable.get_children())
174.
175.                 #table is reloaded into the Treeview
176.                 que = 'SELECT total_price, place_of_purchase, Date_of_Purchase
FROM RECEIPTS'
177.                 tblRows = runQ(que)
178.
179.                 for row in tblRows:
180.                     sqTable.insert('', 'end', values=(row[0], row[1], row[2]))
181.                 gui.destroy()
182.                 messagebox.showinfo("Success", "Record has been updated")
183.             except:
184.                 messagebox.showerror("Error", "Please input the date in format
dd/mm/yy")
185.
186.
187.
188.         #delete and edit buttons
```

```
189.
190.
191.     def createTables():
192.         global sqTable
193.         sqTable = ttk.Treeview(height = 20, columns = ('Total Price',
194.             'Location', 'Date'))
195.
196.         #specified one less as treeview uses #0
197.         sqTable.grid(row = 5, column = 5, columnspan = 7)
198.         sqTable.place(relx = 0.12, rely = 0.2)
199.         sqTable.heading('#0', text = '', anchor = CENTER)
200.         sqTable.column('#0', stretch = NO, minwidth = 0, width = 0)
201.         sqTable.heading('#1', text = 'Date Purchased', anchor = CENTER)
202.         sqTable.heading('#2', text = 'Total Price', anchor = CENTER)
203.         sqTable.heading('#3', text = 'Location', anchor = CENTER)
204.         que = 'SELECT total_price, place_of_purchase, Date_of_Purchase FROM
205.             RECEIPTS'
206.         getTable(sqTable, que)
207.
208.         global itemTable
209.         #treeview to see database items
210.         itemTable = ttk.Treeview(height = 20, column = ("Item Name", "Item
211.             Price"))
212.         itemTable.grid(row = 5, column = 5, columnspan = 7)
213.         itemTable.place(relx = 0.6, rely = 0.2)
214.         itemTable.heading('#0', text = '', anchor = CENTER)
215.         itemTable.column('#0', stretch = NO, minwidth = 0, width = 0)
216.         itemTable.heading('#1', text = 'Item Name', anchor = CENTER)
217.         itemTable.heading('#2', text = 'Item Price', anchor = CENTER)
218.         query = 'SELECT item_purchased, item_price FROM ITEM'
219.         getTable(itemTable, query)
220.
221.     createTables()
222.
223.     Button(text = 'Delete Record', font = 'Adam', command =
224.         deleteRecord).place(relx = 0.6, rely = 0.81)
225.     Button(text = 'Edit Record', font = 'Adam', command =
226.         editReceipt).place(relx = 0.3, rely = 0.81)
```

```
227.         root.mainloop()
```

## GenerateReport.py

```
1. from tkinter import *
2. import tkinter as tk
3. from PIL import ImageTk, Image
4. from tkinter import ttk
5. import sys
6. import os
7. import matplotlib.pyplot as plt
8. import matplotlib.dates as mdates
9. from matplotlib import style
10. import sqlite3 as sq
11. import datetime
12. import time
13. from tkinter import messagebox
14. from fpdf import FPDF
15.
16.
17. root = Tk()
18. root.title("Lucid - Generate Report")
19. root.configure(background = '#ffffff')
20. root.config(height = 1600, width = 800)
21. root.state('zoomed')
22.
23. os.chdir('C:/Users/zubi_/Desktop/[project_skyfall]/Programming/GUI/Current')
24.
25. plt.style.use('seaborn')
26.
27. def InputData():
28.     root.destroy()
29.     import InputData
30.
31. def EditData():
32.     root.destroy()
33.     import EditData
34.
35. def GenerateReport():
36.     root.destroy()
37.     import GenerateReport
38.
39. inputImage = ImageTk.PhotoImage(Image.open("input.png"))
```

```
40. editImage = ImageTk.PhotoImage(Image.open("edit.png"))
41. genImage = ImageTk.PhotoImage(Image.open("genreport.png"))
42.
43.
44.
45. root1 = Button(root, image = inputImage, command=InputData)
46. root1.place(relx = 0.04, rely = 0, anchor= "n")
47.
48.
49. root2 = Button(root, image = editImage, command=EditData)
50. root2.place(relx = 0.115, rely = 0, anchor = "n")
51.
52.
53. root3 = Button(root, image = genImage, command = GenerateReport, state =
    DISABLED)
54. root3.place(relx = 0.19, rely = 0, anchor = "n")
55.
56.
57. db = sq.connect('Clients and Receipts.db')
58. c = db.cursor()
59.
60. #variables for tkinter
61.
62. startST = StringVar()
63. endST = StringVar()
64.
65. startCT = StringVar()
66. endCT = StringVar()
67.
68. startPT = StringVar()
69. endPT = StringVar()
70.
71. dateSpent = StringVar()
72.
73. amount = StringVar()
74. inequality = StringVar()
75.
76. start = StringVar()
77. end = StringVar()
78.
79. #=====Sales Time Graph=====#
80.
81. def salesTimeGraph(start, end):
```



```
82.     try:
83.         #creates unix dates for dates given
84.         unixStart = time.mktime(datetime.datetime.strptime(start,
            "%d/%m/%y").timetuple())
85.         unixEnd = time.mktime(datetime.datetime.strptime(end,
            "%d/%m/%y").timetuple())
86.
87.         query = '''SELECT Date_Of_Sale, Revenue, unix_date_sales FROM
            SALES_MADE ORDER BY unix_date_sales ASC'''
88.         #orders records using unix date
89.         c.execute(query)
90.         dates = []
91.         revenue = []
92.         for row in c.fetchall():
93.             dates.append(row[0])
94.             revenue.append(row[1])
95.             unixDates.append(row[2])
96.             #places each column into a list from the tuple
97.
98.         finalDates = []
99.         #finds dates in range given by user
100.        for i in range(len(unixDates)):
101.            if unixDates[i] >= unixStart and unixDates[i] <= unixEnd:
102.                finalDates.append(unixDates[i])
103.
104.            finalRevenue = []
105.            #finds prices with the same index as dates for plotting
106.            for i in range(len(finalDates)):
107.                a = unixDates.index(finalDates[i])
108.                finalRevenue.append(revenue[a])
109.                obj = datetime.datetime.fromtimestamp(finalDates[i])
110.                finalDates[i] = obj
111.
112.            #plots the graph with dates and revenue
113.            plt.plot_date(finalDates, finalRevenue, '-')
114.            plt.show()
115.        except:
116.            messagebox.showerror("Error", "Please input your dates in format
            dd/mm/yy")
117.            #error given if the unixdates fail - means the date is not in
            correct format
118.
119.        #fields for user to input
```

```
120.         Label(root, text = 'Graph for sales / time', font = ('Adam',
121.             15)).place(relx = 0.05, rely = 0.1)
122.         Label(root, text = 'Start Date', font = 'Adam').place(relx = 0.05, rely =
123.             0.2)
124.         SalesStartDate = Entry(root, textvariable = startST).place(relx = 0.05,
125.             rely = 0.25)
126.         Label(root, text = 'End Date', font = 'Adam').place(relx = 0.05, rely =
127.             0.3)
128.         SalesEndDate = Entry(root, textvariable = endST).place(relx = 0.05, rely
129.             = 0.35)
130.         #calls subroutine and passes on dates specified in fields
131.         Button(root, text = "Generate", font = "Adam", command = lambda:
132.             salesTimeGraph(startST.get(), endST.get())).place(relx = 0.1, rely = 0.4)
133.
134.         #=====Costs Time Graph=====#
135.         def costsTimeGraph(start, end):
136.             db = sq.connect('Clients and Receipts[NEW].db')
137.             c = db.cursor()
138.             #converts dates into unix
139.             unixStart = time.mktime(datetime.datetime.strptime(start,
140.                 "%d/%m/%y").timetuple())
141.             unixEnd = time.mktime(datetime.datetime.strptime(end,
142.                 "%d/%m/%y").timetuple())
143.
144.             query = '''SELECT total_price, Date_of_Purchase, unix_date_purchase
145.                 FROM RECEIPTS ORDER BY unix_date_purchase ASC'''
146.
147.             #orders columns using unix
148.             #gets dates, costs and unix dates from columns
149.             c.execute(query)
150.             dates = []
151.             costs = []
152.             unixDates = []
153.             for row in c.fetchall():
154.                 costs.append(row[0])
155.                 dates.append(row[1])
156.                 unixDates.append(row[2])
157.
158.             finalDates = []
159.             #finds dates in range given by user
```

```

154.         for i in range(len(unixDates)):
155.             if unixDates[i] >= unixStart and unixDates[i] <= unixEnd:
156.                 finalDates.append(unixDates[i])
157.
158.         finalCosts = []
159.         #gets the same index costs as dates for plotting
160.         for i in range(len(finalDates)):
161.             a = unixDates.index(finalDates[i])
162.             finalCosts.append(costs[a])
163.             obj = datetime.datetime.fromtimestamp(finalDates[i])
164.             finalDates[i] = obj
165.
166.         #plots graphs
167.         plt.plot_date(finalDates, finalCosts, '-')
168.         plt.show()
169.
170.         Label(root, text = 'Graph for costs / time', font = ('Adam',
171.             15)).place(relx = 0.05, rely = 0.6)
172.         Label(root, text = 'Start Date', font = 'Adam').place(relx = 0.05, rely =
173.             0.7)
174.         CostsStartDate = Entry(root, textvariable = startCT).place(relx = 0.05,
175.             rely = 0.75)
176.         Label(root, text = 'End Date', font = 'Adam').place(relx = 0.05, rely =
177.             0.8)
178.         CostsEndDate = Entry(root, textvariable = endCT).place(relx = 0.05, rely
179.             = 0.85)
180.
181.         Button(root, text = "Generate", font = "Adam", command = lambda:
182.             costsTimeGraph(startCT.get(), endCT.get())).place(relx = 0.1, rely = 0.9)
183.
184.         #####Profits Time Graph#####
185.
186.         def profitsGraph(start, end):
187.             unixStart = time.mktime(datetime.datetime.strptime(start,
188.                 "%d/%m/%y").timetuple())
189.             unixEnd = time.mktime(datetime.datetime.strptime(end,
190.                 "%d/%m/%y").timetuple())
191.
192.             query = '''SELECT total_price, Date_of_Purchase FROM RECEIPTS ORDER
193.                 BY unix_date_purchase ASC'''
194.             c.execute(query)
195.             datesCosts = []

```

```
188.         costs = []
189.         for row in c.fetchall():
190.             costs.append(row[0])
191.             datesCosts.append(row[1])
192.
193.
194.         query = '''SELECT Date_Of_Sale, Revenue FROM SALES_MADE ORDER BY
        unix_date_sales ASC'''
195.         c.execute(query)
196.         datesSales = []
197.         revenue = []
198.         for row in c.fetchall():
199.             datesSales.append(row[0])
200.             revenue.append(row[1])
201.
202.
203.
204.         for i in range (len(datesCosts)):
205.             if costs.index(datesCosts[i]):
206.                 costIndex = salesCosts.index(datesCosts[i])
207.                 finalCosts.append(costs[costIndex])
208.         #profits can be between dates
209.         plt.plot_date(datesSales, revenue, '-')
210.         plt.plot_date(datesCosts, costs, '-')
211.         plt.show()
212.
213.         Label(root, text = 'Graph for Profts', font = ('Adam', 15)).place(relx =
        0.35, rely = 0.1)
214.         Label(root, text = 'Start Date', font = 'Adam').place(relx = 0.35, rely =
        0.2)
215.         ProfitsStartDate = Entry(root, textvariable = startPT).place(relx = 0.35,
        rely = 0.25)
216.         Label(root, text = 'End Date', font = 'Adam').place(relx = 0.35, rely =
        0.3)
217.         ProfitsEndDate = Entry(root, textvariable = endPT).place(relx = 0.35,
        rely = 0.35)
218.
219.         Button(root, text = "Generate", font = "Adam", command = lambda:
        profitsGraph(startPT.get(), endPT.get())).place(relx = 0.4, rely = 0.4)
220.
221.
222.
223.         #=====Amount Spent on day X=====
```

```
224.
225.     def DateSpentX(date):
226.         try: #checks to see if the user has entered the correct date
227.             unixDate = time.mktime(datetime.datetime.strptime(date,
228.                 "%d/%m/%y").timetuple())
229.             query = '''SELECT total_price FROM RECEIPTS WHERE
230.                 Date_of_Purchase = ?'''
231.             c.execute(query, (date,))
232.             spent = []
233.             for i in c.fetchall():
234.                 spent.append(i[0])
235.                 #gets all the prices of the date the client may have spent
236.             total = 0
237.
238.             for i in range(len(spent)):
239.                 total += spent[i]
240.                 #adds the value up
241.
242.             message = ("You have spent a total of: £"+ str(total))
243.             messagebox.showinfo("Success",message)
244.             #presented to user
245.         except:
246.             messagebox.showerror("Error", "Date not found")
247.
248.
249.         #field for input
250.         Label(root, text = "Reports:", font = ("Adam", 25)).place(relx = 0.75, rely
251.             = 0.05)
252.         Label(root, text = "Money spent on day:", font = 'Adam').place(relx =
253.             0.6, rely = 0.2)
254.         DateSpent = Entry(root, textvariable = dateSpent).place(relx = 0.6, rely
255.             = 0.25)
256.
257.         Button(root, text = "Generate", font = 'Adam', command = lambda:
258.             DateSpentX(dateSpent.get())).place(relx = 0.65, rely = 0.3)
259.
260.         #=====Amount spent More/Less Than=====#
261.         def AmountSpent(amount, inequality):
262.             query = '''SELECT Date_of_Purchase, total_price FROM RECEIPTS ORDER
263.                 BY unix_date_purchase ASC'''
264.             #ordered by unix
```

```
260.         c.execute(query)
261.
262.         spent = []
263.         dates = []
264.         for i in c.fetchall():
265.             dates.append(i[0])
266.             spent.append(i[1])
267.             #gets all dates and amount spent on those days
268.
269.         totalSpent = []
270.         amountDate = []
271.         if inequality == "More Than":
272.             report = FPDF()
273.
274.             #pdf file is made
275.             report.add_page()
276.             report.set_font("Arial", size = 15)
277.             report_line1 = "Days spent more than: £"+amount
278.             report.cell(200, 10, txt = report_line1,
279.                 ln = 1, align = 'C')
280.             for i in range(len(spent)): #looks for lentgh of spent
281.                 if float(amount) < float(spent[i]):
282.                     totalSpent.append(spent[i])
283.                     amountDate.append(dates[i])
284.                     #appends each item to the list
285.
286.             report.cell(200, 10, txt = "Date: "+str(amountDate[i]), ln =
(2 + i), align = 'L')
287.             #for each new receipt, it is
288.             #placed on a new line
289.             report.cell(200, 10, txt = "Amount: "+str(totalSpent[i]), ln
= (2 + i), align = 'C')
290.
291.             report.output("AmountSpentMoreThanX.pdf", 'F')
292.             messagebox.showinfo("Success", "Please check your folder for the
report")
293.
294.         elif inequality == "Less Than":
295.             report = FPDF()
296.             report.add_page()
297.             report.set_font("Arial", size = 15)
298.             report_line1 = "Days spent more than: £"+amount
299.             report.cell(200, 10, txt = report_line1,
```

```
300.         ln = 1, align = 'C')
301.     for i in range(len(spent)):
302.         if float(amount) > float(spent[i]):
303.             totalSpent.append(spent[i])
304.             amountDate.append(dates[i])
305.         #gets all amounts which are less than asked more and the
           allocated dates
306.
307.         report.cell(200, 10, txt = "Date: "+str(amountDate[i]), ln =
           (2 + i), align = 'L')
308.         #for each new receipt, it is
309.         #placed on a new line
310.         report.cell(200, 10, txt = "Amount: "+str(totalSpent[i]), ln
           = (2 + i), align = 'C')
311.
312.         report.output("AmountSpentLessThanX.pdf", 'F')
313.         messagebox.showinfo("Success", "Please check your folder for the
           report")
314.     else:
315.         messagebox.showerror("Error", "Please check your fields")
316.         #if unix fails, inputs are asked for again
317.
318.
319.
320.     #fields for input
321.     Label(root, text = "Amount spent:", font = 'Adam').place(relx = 0.6, rely
           = 0.5)
322.     Inequality = Entry(root, textvariable = inequality)
323.     Inequality.place(relx = 0.6, rely = 0.55)
324.     Label(root, text = "More Than/Less Than", font = 'Adam').place(relx =
           0.6, rely = 0.5)
325.     Amount = Entry(root, textvariable = amount)
326.     Amount.place(relx = 0.6, rely = 0.65)
327.
328.     #calls subroutine and passes on inputs as parameters
329.     Button(root, text = "Generate", font = 'Adam', command = lambda:
           AmountSpent(amount.get(), inequality.get())).place(relx = 0.65, rely = 0.7)
330.
331.     =====Profit Loss Statement=====
332.     def getBalanceSheet(start, end):
333.         #converts into unix dates
334.         unixStart = time.mktime(datetime.datetime.strptime(start,
           "%d/%m/%y").timetuple())
```

```
335.         unixEnd = time.mktime(datetime.datetime.strptime(end,
"%d/%m/%y").timetuple())
336.
337.
338.         c.execute('''SELECT total_price, Date_of_Purchase, unix_date_purchase
FROM RECEIPTS ORDER BY unix_date_purchase ASC''')
339.         costsData = c.fetchall() #has tuples with total price, date, unix in
order
340.         costs = []
341.         costDate = []
342.         unixPurchase = []
343.         for row in costsData:
344.             costs.append(row[0])
345.             costDate.append(row[1])
346.             unixPurchase.append(row[2])
347.         #gets costs, dates and unix date from tuple into their own list
348.
349.
350.
351.         finalDatesCosts = []
352.         finalCosts = []
353.         for i in range(len(unixPurchase)):
354.             if unixPurchase[i] >= unixStart and unixPurchase[i] <= unixEnd:
355.                 finalDatesCosts.append(costDate[i])
356.                 finalCosts.append(costs[i])
357.         #this loop gives the dates that between the user wants the report
between - from the receipts table
358.
359.         c.execute('''SELECT Date_Of_Sale, Revenue, unix_date_sales FROM
SALES_MADE ORDER BY unix_date_sales ASC''')
360.         salesDate = []
361.         sales = []
362.         unixSales = []
363.         for row in c.fetchall():
364.             salesDate.append(row[0])
365.             sales.append(row[1])
366.             unixSales.append(row[2])
367.         #gets costs, dates and unix date from tuple into their own list
368.
369.         finalDatesSales = []
370.         finalSales = []
371.         for i in range(len(unixSales)):
372.             if unixSales[i] >= unixStart and unixSales[i] <= unixEnd:
```



```
373.             finalDatesSales.append(salesDate[i])
374.             finalSales.append(sales[i])
375.             #this loop gives the dates that between the user wants the report
               between - from the sales tables
376.
377.             Revenue = 0
378.             Costs = 0
379.
380.             def nothing():
381.                 #for date in range(len(finalDatesCosts)):
382.                 #     for i in range(12): #number of months in a year
383.                 #         if '0'+str(i+1) == finalDatesCosts[date][3:5] or
               str(i+1) == finalDatesCosts[date][3:5]: #looks for
384.                 #             for a in range(len(finalCosts)):
385.                 #                 Revenue += float(finalCosts[date])
386.                 #             else:
387.                 #                 totalMonths9 = 0
388.                 None
389.
390.             for i in range(len(finalCosts)):
391.                 Revenue += float(finalCosts[i])
392.                 #adds up all of the revenue as a string
393.
394.             for i in range(len(finalSales)):
395.                 Costs += float(finalSales[i])
396.                 #adds up all of the costs
397.
398.             Profits = str(Revenue - Costs)
399.             #calculates profit
400.             print('£' +str(Costs))
401.             print('£'+str(Revenue))
402.             print(Profits)
403.
404.
405.
406.             #places data in a pdf
407.             report = FPDF()
408.             report.add_page()
409.             report.set_font("Arial", size = 15)
410.             report_line1 = "Profit/Loss Statement for dates "+ start+ " - " + end
411.             report.cell(200, 10, txt = report_line1,
412.                 ln = 1, align = 'C')
413.
```

```
414.         # add another cell
415.         report.cell(200, 10, txt = "Statement for Yunus Saleh",ln = 2, align
            = 'L')
416.         report.cell(200, 10, txt = "Statement for Yunus Saleh - Tailoring",
            ln = 3, align = 'L')
417.         report.cell(200, 10, txt = "Total Revenue £:", ln = 4, align = 'L')
418.         report.cell(200, 10, txt = str(Revenue), ln = 4, align = 'R')
419.         report.cell(200, 10, txt = "Total Costs £:", ln = 5, align = 'L')
420.         report.cell(200, 10, txt = str(Costs), ln = 5, align = 'R')
421.         report.cell(200, 10, txt = "Total Proits £:", ln = 6, align = 'L')
422.         report.cell(200, 10, txt = Profits, ln = 6, align = 'R')
423.
424.         report.output("Profit/Loss.pdf", 'F')
425.
426.
427.         #fields to input
428.         Label(root, text = 'Graph for Cashflow', font = ('Adam', 15)).place(relx
            = 0.35, rely = 0.6)
429.         Label(root, text = 'Start Date', font = 'Adam').place(relx = 0.35, rely =
            0.7)
430.         CostsStartDate = Entry(root, textvariable = start).place(relx = 0.35,
            rely = 0.75)
431.         Label(root, text = 'End Date', font = 'Adam').place(relx = 0.35, rely =
            0.8)
432.         CostsEndDate = Entry(root, textvariable = end).place(relx = 0.35, rely =
            0.85)
433.
434.         #calls subroutine by passing parameters from fields
435.         Button(root, text = "Generate", font = "Adam", command = lambda:
            getBalanceSheet(str(start.get()), str(end.get()))).place(relx = 0.35, rely =
            0.9)
436.
437.
438.
439.         root.mainloop()
```

## Testing

Test Number	Test Item	Example	Expected Outcome		Changes made/reasons for errors	Success?
(1)	<b>Log In Page</b>		Once the program is initialised, the Log in GUI should be displayed with necessary fields and button.	Y	Denied access	
(1.2)	Username /Password		If the wrong username/password is entered, an error should be given	Y	Access was denied as expected	
(1.3)	Password	*****	The password should not be visible to be seen.	Y	Password was shown as is seen in the example in the example	
(1.4)	Username, Password	Zubaid12, lUcid%Rec Ieptz	If the credentials are correct, a message should notify this. and the choices page should pop up if the user has the correct credentials.	N	Denied Access as username was written with an extra space, so access was denied as it did not match the database. Username is now updated	Y
(2)	Choices Page		The choices page should pop up if the user has the correct credentials.		Choices page popped up after dialog box.	
(3)	<b>InputData, Edit, Report</b>	Click between them when starting the program	If the user chooses to use any of these, they can rotate between them	N	Program closed as the subroutine pointed to the folder with the wrong name. These names were updated and now work correctly.	Y
(3.1)		Click between them mid processes	If a process has started, then the user can still change between choices		Works correctly.	

<b>(4)</b> <b>Input Data</b>	Receipt Images	Choose images to scan	Allow the user to choose images via the file explorer.		File explorer opened allowing user to choose images	
<b>(4.1)</b>		Empty Box	Return an error if there are no files in the box		Returned error as expected	
<b>(4.2)</b>		Correct File Format	Make sure the files that can be selected are correct in format			
<b>(4.3)</b>		Number of files selected	Allows the user to select more than one file of the same type and place this into the box.		Allows the user to select multiple "*.png" or "*.jpg" files together.	
<b>(4.4)</b>		Tesseract	Tesseract should scan the images for the costs data items needed	<b>N</b>	Database was named wrong in the InputData file.	<b>Y</b>
					placeIndex may not always be defined so an error was given for one of the receipts. // This was fixed by giving a value to placeIndex, the user can edit this in the edit area.	<b>Y</b>
					Only the first receipt in the whole list was scanned.  //the storing procedure ran once, and the success message stopped the program for other receipts.  This was fixed by placing success message at the end at the end of the tesseract function.	<b>Y</b>

				Error was given when the program tried to turn a string containing characters into an integer.  //fixed using exception handling	Y
(4.5)		Data points	Data points should be stored onto the correct table on the database	Only stores the first receipt in the list.  //Indentation was not correct. Now stores each receipt in the database	Y
(5)	Input Costs	KFC, £5.99, 12/09/18	The corresponding values should be able to be input in the boxes given. If they are empty, they should not update database. If all values are given(in the correct format) the database should update.		
(5.1)		Input Box	Input boxes should display to the user when button is clicked		
(5.2)		Input Fields	When the fields are empty and the boxes are clicked, an error should display		
(5.3)		Input Fields e.g. date should be `dd/mm/yy`	Fields should only take the correct format as required		
(5.4)		Button Clicked	Should store the values into the database		

<b>(5.5)</b>		ReceiptId	Should increment by one as values are added			
<b>(6)</b>	<b>Input Sales</b>	Hat	The corresponding values should be all added to a database. Fields should not be empty.	<b>N</b>	Did not update the database with the correct inputs.	<b>Y</b>
<b>(6.1)</b>	Item Name, Date, Price	Hat, 21/03/19, 4.50	The value should be able to be placed into entry boxes.	<b>N</b>	//the query was not written correctly when updating in SQL	<b>Y</b>
<b>(6.2)</b>		Update Button	Button should run function which would update given values into the program.	<b>N</b>		<b>Y</b>
<b>(6.3)</b>	Update Button		Button should output an error message if the dates are not in the correct format.			

## Changes made to code

### Test 4.4 - Input Data

```
def storeToTes(date, subtotal, location, item, itemPrice):
    db = sq.connect('Clients and Receipts[NEW].db')
    c = db.cursor()

    try:
        unixDate = time.mktime(datetime.datetime.strptime(date, "%d/%m/%y").timetuple())
        #date = datetime.datetime.fromtimestamp(unixDate)
        query = ''' INSERT INTO RECEIPTS
                    (client_id ,Date_of_Purchase, unix_date_purchase, total_price, place_of_purchase)
                    VALUES
                    (?, ?, ?, ?, ?)
                '''
```

Before

```
def storeToTes(date, subtotal, location, item, itemPrice):
    db = sq.connect('Clients and Receipts.db')
    c = db.cursor()

    try:
        unixDate = time.mktime(datetime.datetime.strptime(date, "%d/%m/%y").timetuple())
        #date = datetime.datetime.fromtimestamp(unixDate)
        query = ''' INSERT INTO RECEIPTS
                    (client_id ,Date_of_Purchase, unix_date_purchase, total_price, place_of_purchase)
                    VALUES
                    (?, ?, ?, ?, ?)
                '''
```

After

### Error 2 - PlaceIndex

```
for i in range(len(values)):
    if findSubstring(values[i], 'Road') == True or findSubstring(values[i], 'road') == True or findSubst
        placeIndex = i
```

Before

```
for i in range(len(values)):
    if findSubstring(values[i], 'Road') == True or findSubstring(values[i],
        placeIndex = i
    else:
        placeIndex = 1
```

After

## Error 4 - Integer String

```
if int(values[placeIndex - 2]) == int(values[placeIndex - 2]):  
    address = values[placeIndex - 2] , values[placeIndex - 1] , values[placeIndex]  
elif int(values[placeIndex + 1]):  
    address = values[placeIndex + 1]  
else:  
    None
```

Before

```
try:  
    if int(values[placeIndex - 2]) == int(values[placeIndex - 2]):  
        address = values[placeIndex - 2] , values[placeIndex - 1] , values[placeIndex]  
    elif int(values[placeIndex + 1]):  
        address = values[placeIndex + 1]  
    else:  
        None  
except:  
    address = 1
```

After

## Test 6, 6.2, 6.2 - Input Sales

```
def storeToSales(gui, Date, Price, Item):  
    db = sq.connect('Clients and Receipts.db')  
    c = db.cursor()  
  
    try: #exception handling to see if a date exists  
        unixDate = time.mktime(datetime.datetime.strptime(Date, "%d/%m/%y").timetuple())  
        query = '''INSERT INTO SALES_MADE  
        (client_id, Date_Of_Sale, unix_date_sales, Revenue, ItemSold)  
        VALUES  
        (?, ?, ?, ?, ?)  
        WHERE (place_of_purchase) = 'none' '''  
        sale = (client_id, str(Date), float(unixDate), float(Price), str(Item))
```

Before

```
def storeToSales(gui, Date, Price, Item):  
    db = sq.connect('Clients and Receipts.db')  
    c = db.cursor()  
  
    try: #exception handling to see if a date exists  
        unixDate = time.mktime(datetime.datetime.strptime(Date, "%d/%m/%y").timetuple())  
        query = '''INSERT INTO SALES_MADE  
        (client_id, Date_Of_Sale, unix_date_sales, Revenue, ItemSold)  
        VALUES  
        (?, ?, ?, ?, ?)  
        '''  
        sale = (client_id, str(Date), float(unixDate), float(Price), str(Item))
```

After



## Evaluation

The project has been completed and does what it is supposed to do. It follows the initial design and analysis and therefore it is time to check how successful the solution was. To do this, I have decided to compare how I have done with regards to my objects, how the client finds the program, and how I may have executed the solution in better ways.

### Objective Comparison

Objective Number	Objective	Execution/Improvements?
(1)	Program must have a login system This must be robust and only allow access if the username and password are stored in the database. This will add security. Data of the correct client must be displayed to that client. E.g. if there were a second client, client one will not be able to access the data of client two.	Program has a robust log in system which works on SQL and works effectively. It does not allow the user in if they do not have correct details. To make this aspect of the program better, I would have placed a 'forgot password' section. This would have added to the robustness of the system and made it more secure. If the password was compromised the user would have been able to change it. Due to time, I was also unable to implement a working hashing algorithm for my password, so this would have been a part of the program too.
(2)	Program must allow choice for the user between Input/Edit/Reports	This section of the program worked perfectly. The user was able to choose what they wanted to do without any problem. This meant that the user could choose what to do and not have to follow a sequence. To improve this, I could have kept the three sections on the same window and allowed them to run simultaneously in the program. This would have made my program more dynamic than it already is.
(3)	Tesseract must work to store receipt data into the database. By storing OCR data to a dictionary and using regular expressions to identify the fields needed to store into the database.	This section of the program worked effectively. Tesseract was able to provide a dictionary with the details I have mentioned in the analysis effortlessly. Identifying the different components of the receipt using regex could have been performed better in my opinion as the success rate for a complete receipt was not very high. For example, the items and their price on the receipt did not scan very well. However, these were not needed for the

		<p>final reports that were generated at the end of the program. This meant that the implementation was enough for the services that my client required. TO make this section of the program better, I would have implemented some use for the items and allowed the user to do something with the item data. This would have added more functionality for the program. The Tesseract engine did also take a few seconds to work – although this may be due to the performance of the computer I programmed on, which has a poor GPU.</p>
<b>(4)</b>	<p>Manually inputting data; This can be sales or receipt data. Where there is receipt data, the user must be able to enter the fields, and the date must be converted to a Unix date and stored in the database. If the user wants to store a specific item that exists on the receipt, they must be able to add items. The items and receipts must share the same receipt_id.</p>	<p>This section of the program was very successful. If the user wanted to input sales/input some handwritten receipts they were able to input them manually. This was a matter a few entry fields and SQL statements to add these to the receipts. Th program was able to make sure that the date was given in the format that was needed to calculate a Unix Date. To make this section better I would have written this using OOP due to the repetitive nature of the program. This would have been more useful and worked at a higher performance.</p>
<b>(5)</b>	<p>Editing Data; The user must be able to edit the tables that are stored in the database; these are the receipt and item tables. The data in the GUI must be brought from the database and as they are updated/deleted, the values in the database should change accordingly. This must be done in an efficient manner.</p>	<p>This section of the program also worked very well. The program was able to retrieve the tables from the Database and present them in a Treeview table. This meant that the user was fully able to interact with the fields as they wished. If the user had chosen to edit/delete fields this was not a problem and the Treeview would update instantly. This would mean that there was no lag in the process, which made it simplistic to use. However, if I had more time, I would have allowed the user to edit the Items table and tell them which items link with which receipts. This would have been useful for the client to know about.</p>
<b>(6)</b>	<p>Reports Generation; The user must be able to generate reports according to the one that they choose. If the user chooses to generate a graph, then the graph must be generated based on the</p>	<p>Tis section of the program was also a successful aspect of the program. The user was able to choose between which report/graph they want to use, and between which dates they would want</p>

	<p>dates that are given. This should pop up and allow the user to interact with it.</p>	<p>these. This is a very useful part of the program as it uses all the data that is fed by the client to show how he is doing in a visual manner. This was a problem for Mr Saleh as he could not 'see' how his program was overall doing. This part of the program allowed him to see that and will hopefully contribute to a healthier business future for him. To make this section of the program better I would have integrated the graphs into the GUI instead of the graphs popping up after you input the dates. This would have made the program more dynamic and better to see visually. If there was an option to see all sales/all costs, this would have been nice too.</p>
--	---	--

## Client Feedback

Objective Number	Objective	Improvements?
(1)	<p>Program must have a login system This must be robust and only allow access if the username and password are stored in the database. This will add security. Data of the correct client must be displayed to that client. E.g. if there were a second client, client one will not be able to access the data of client two.</p>	<p>"Was easy to use and access. Lucid opened relatively quickly and I was able to put in my given username and password. After I logged in, I was greeted and was able to choose what I wanted to do, although if I was to forget my password/username I would have to contact you. Other than that, it is perfect."</p>
(2)	<p>Program must allow choice for the user between Input/Edit/Reports</p>	<p>"Nice to see I was greeted, and I was able to choose which process I wanted to use. No improvements."</p>
(3)	<p>Tesseract must work to store receipt data into the database. By storing OCR data to a dictionary and using regular expressions to identify the fields needed to store into the database.</p>	<p>"Program has a simple easy-to-follow GUI. Nothing fancy, and not too simple. I was able to manoeuvre the program when I arrived at the input page. Here, I was able to place pictures I took/receipts I scanned using my scanner. It was simple to select these files for Lucid to scan. Although – it was a bit annoying when I had to select png and jpg files separately. It would have also been nice to use pdf files also. Tesseract overall worked okay but not perfect. I found myself editing these in the</p>

		Edit page, but it did get the simple fields such as total price, names and dates correct some of the time – quicker than counting these by hand so no complains”
<b>(4)</b>	Manually inputting data; This can be sales or receipt data. Where there is receipt data, the user must be able to enter the fields, and the date must be converted to a Unix date and stored in the database. If the user wants to store a specific item that exists on the receipt, they must be able to add items. The items and receipts must share the same receipt_id.	“This was an easy part of the program. It was very useful to me to input these values when I had made a sale as it was reassuring to me that I did not have to calculate this at the end of a long term. It helped me keep track of sales and helped with my organisation. Inputting receipt data is also very useful as they keep my costs ready too. It would have been a nice option if I didn’t have to input the date if sale was made today/bought something today. This would have cut down some time.”
<b>(5)</b>	Editing Data; The user must be able to edit the tables that are stored in the database; these are the receipt and item tables. The data in the GUI must be brought from the database and as they are updated/deleted, the values in the database should change accordingly. This must be done in an efficient manner.	“I was able to edit the tables without problem. Where there were errors in the Lucid scanning, I was able to change them without hesitation. Although, the items table was there for just show. I was not able to interact with it/know which receipt it was linked to. This is unimportant however as the main thing I need is the receipt table.”
<b>(6)</b>	Reports Generation; The user must be able to generate reports according to the one that they choose. If the user chooses to generate a graph, then the graph must be generated based on the dates that are given. This should pop up and allow the user to interact with it.	“Generating my profit/loss sheet was effortless. I was able to just input the dates between I wanted the reports and they were just given to me without any problem. This meant that if I needed an emergency one as soon as possible, I didn’t have to do all the calculations manually. We didn’t discuss the need for the other reports, but I have found them to also be very useful. Seeing how much I am making and buying visually has helped me a lot. This has made me aware of the decisions that I am making within my business and I hope that these decisions lead to better things.”

## Self-Evaluation

I believe that the project has been successful in that it provided my client with what he needs and solved the problem he was facing when I interviewed him at the beginning of the project. This means that I was successful, but I think that there were things I could have done to make this more successful.

### Planning

As a programmer, it is my duty to meet deadlines and make sure I know how I am doing in the project. And when it comes to my planning, I believe I could have done a lot better than I did. I did not have a coherent plan to follow with dates and progress sheets which meant that I did not know how much there was left to do and how much I had left to finish the tasks. This led to some things not working perfectly in my program which – were minor – but could have been better.

I could have used a Gantt chart which would show how much time I would spend on a certain part of the program along with each task broken down. This would have been a good practise and allowed me to meet deadlines with complete solutions. I found myself making handmade lists and ticking these off at the end of the project because I had to make sure I had done everything that I needed to do. This was poor organisation on my behalf, and I would have done this differently if I were to do this project again.

## References

### Harvard Reference Table

Source Name	Source Type	Source Link	Source used	Date Accessed	Comments
<b>Quora</b>	Forum	<a href="https://www.quora.com/What-is-SQL-and-why-is-it-important">https://www.quora.com/What-is-SQL-and-why-is-it-important</a>	SQL Statements	06/10/19	-
<b>Sitepoint</b>	Learning tool	<a href="https://www.sitepoint.com/getting-started-sqlite3-basic-commands/">https://www.sitepoint.com/getting-started-sqlite3-basic-commands/</a>	SQL Statements	06/10/19	-
<b>Legalscans</b>	Website	<a href="http://www.legalscans.com/ocr.html">http://www.legalscans.com/ocr.html</a>	OCR API	18/10/19	
<b>Quora</b>	Forum	<a href="https://www.quora.com/How-does-an-API-work-Where-can-I-learn-about-APIs-Are-there-any-good-books-on-the-subject-How-do-I-make-my-own">https://www.quora.com/How-does-an-API-work-Where-can-I-learn-about-APIs-Are-there-any-good-books-on-the-subject-How-do-I-make-my-own</a>	OCR API		