A computer screen shot of a computer

Description automatically generated

Smart Receipt Management and Extraction

Final Report

M00843707

Damien Maujean

28 APRIL 2024

Supervise by honorable

Nivede Issur

[Table of figure 4](#_Toc165136703)

[1 Abstract 6](#_Toc165136704)

[2 Introduction 6](#_Toc165136705)

[2.1 Problem statement 6](#_Toc165136706)

[2.2 Background 7](#_Toc165136707)

[2.3 Aims 7](#_Toc165136708)

[2.4 Objectives 8](#_Toc165136709)

[3 Literature review 8](#_Toc165136710)

[3.1 Receipt Extractor 8](#_Toc165136711)

[3.1.1 Paper 1 - Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms. 8](#_Toc165136712)

[3.1.2 Paper 2 - Information Extraction from Scanned Invoices using Machine Learning, OCR, and Spatial Feature Mapping Techniques 11](#_Toc165136713)

[3.1.3 Paper 3 - Computer Vision for Document Image Analysis and Text Extraction 12](#_Toc165136714)

[3.2 Mobile Application 14](#_Toc165136715)

[3.2.1 Paper 4 - React Native vs Flutter, cross-platform mobile application frameworks. 14](#_Toc165136716)

[4 Requirements specification 16](#_Toc165136717)

[4.1 Receipt Section Detection 16](#_Toc165136718)

[4.2 Text Extraction 16](#_Toc165136719)

[4.3 Format and Classify text extraction. 16](#_Toc165136720)

[4.5 User Interface 17](#_Toc165136721)

[5 Design 17](#_Toc165136722)

[5.1 Server 20](#_Toc165136723)

[5.1.1 Receipt extraction feature. 20](#_Toc165136724)

[5.1.2 User Correction Feature. 21](#_Toc165136725)

[5.1.3 Server Communication 21](#_Toc165136726)

[5.1.4 Database 22](#_Toc165136727)

[5.2 Mobile Application 23](#_Toc165136728)

[5.2.1 Receipt upload for receipt extraction. 23](#_Toc165136729)

[5.2.2 List Receipt Historical 25](#_Toc165136730)

[5.2.3 Analytics 25](#_Toc165136731)

[6 Implementation 26](#_Toc165136732)

[6.1 Server 26](#_Toc165136733)

[6.1.1 Docker Usage 29](#_Toc165136734)

[6.1.2 Makefile 29](#_Toc165136735)

[6.1.3 Receipt Section Detection Model (YOLO) 31](#_Toc165136736)

[6.1.3.1 Dataset 33](#_Toc165136737)

[6.1.3.2 Training Process 34](#_Toc165136738)

[6.1.4 Classification/Formatting 35](#_Toc165136739)

[6.2 Mobile Application 37](#_Toc165136740)

[6.2.1 Classes 37](#_Toc165136741)

[6.2.2 Widgets 39](#_Toc165136742)

[6.2.3 Pages 50](#_Toc165136743)

[7 Testing 53](#_Toc165136744)

[7.1 Application 53](#_Toc165136745)

[7.1.1 Classes 54](#_Toc165136746)

[7.1.2 Widgets 54](#_Toc165136747)

[7.1.3 Pages 55](#_Toc165136748)

[7.2 Server 56](#_Toc165136749)

[7.2.1 Endpoint (FastAPI App) 56](#_Toc165136750)

[7.2.2 Receipt Section Detection Model. 57](#_Toc165136751)

[8 Results & Analysis 60](#_Toc165136752)

[8.1 Results 60](#_Toc165136753)

[8.2 Analysis 60](#_Toc165136754)

[9 Conclusion 62](#_Toc165136755)

[Reference 66](#_Toc165136756)

[Appendix 67](#_Toc165136757)

# Table of figure

[Figure 1 - System Design Diagram 18](#_Toc165195070)

[Figure 2 - System Class Diagram 19](#_Toc165195071)

[Figure 3 - Receipt Extraction Feature output format. 21](#_Toc165195072)

[Figure 4 - Database Schema 22](#_Toc165195073)

[Figure 5 - Receipt Extraction Feature Sequence Diagram 24](#_Toc165195074)

[Figure 6 - System Implementation Schema 26](#_Toc165195075)

[Figure 7 - Server FastAPI Code 29](#_Toc165195076)

[Figure 8 - Makefile Code 31](#_Toc165195077)

[Figure 9 - prediction model area with prediction percentage. 32](#_Toc165195078)

[Figure 10 - Extract Receipt Section Code 33](#_Toc165195079)

[Figure 11 - Dataset Distribution 33](#_Toc165195080)

[Figure 12 - Model Training code used on Google Collab 34](#_Toc165195081)

[Figure 13 - Chatgpt classifying and formatting prompt. 36](#_Toc165195082)

[Figure 14 - Data Services Class Code 38](#_Toc165195083)

[Figure 15 - Receipt class parameters. 38](#_Toc165195084)

[Figure 16 - Receipt Class Code 39](#_Toc165195085)

[Figure 17 - Navigation bar. 40](#_Toc165195086)

[Figure 18 - Navigation Bar Widget Code 41](#_Toc165195087)

[Figure 19 - Camera Widget 41](#_Toc165195088)

[Figure 20 - Camera Widget Code 43](#_Toc165195089)

[Figure 21 - Camera Roll Widget 43](#_Toc165195090)

[Figure 22 - Camera Roll Widget Code 44](#_Toc165195091)

[Figure 23 - Popup Widget 44](#_Toc165195092)

[Figure 24 - Popup Widget Code 49](#_Toc165195093)

[Figure 25 - Analytics Widgets States 50](#_Toc165195094)

[Figure 26 - Camera Page States 51](#_Toc165195095)

[Figure 27 - List Pages States 52](#_Toc165195096)

[Figure 28 - Graph Page States 52](#_Toc165195097)

[Figure 29 - Waterfall Framework Steps 53](#_Toc165195098)

[Figure 30 - Flutter testing result logs. 54](#_Toc165195099)

[Figure 31 - Server test result logs. 57](#_Toc165195100)

[Figure 32- Old Brother Restaurant Receipt 61](#_Toc165195101)

[Table 1 - Advantage and Limitation of technology used in "Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms" written by Odd and Theologou 10](#_Toc165195102)

[Table 2 - Advantage and Limitation of technologies used in "Information Extraction from Scanned Invoices using Machine Learning, OCR and Spatial Feature Mapping Techniques" written by Darsha. 12](#_Toc165195103)

[Table 3- Advantage and Limitation of technologies used in "Computer Vision for Document Image Analysis and Text Extraction" written by Benchekrou. 14](#_Toc165195104)

[Table 4 - Advantage and limitation of Flutter and React Native from "React Native vs Flutter, cross-platform mobile application frameworks" written by Wu. 16](#_Toc165195105)

[Table 5 - Receipt Section Detection Model accuracy result` 60](#_Toc165195106)

[Appendix 1 - Camera Page code 70](#_Toc165194984)

[Appendix 2 - Graph Page Code 74](#_Toc165194985)

[Appendix 3 - List Page Code 76](#_Toc165194986)

[Appendix 4 - Meeting Log Part 1 76](#_Toc165194987)

[Appendix 5 - Meeting Log Part 2 77](#_Toc165194988)

# 1 Abstract

This project focus on the error-prone and time-consuming process of manual receipt logging and management within company or personal accounting management. It identifies the problem to solve as the need to assist and simplify the logging and organization of receipt data to enhance efficiently in financing process. To solve this problem a mobile application working with a server was developed, using in the server advance model and Optical Character Recognition (OCR) technologies to accurately extract and categorize information from digitalized receipts. The application, designed using Flutter framework, allow the user to upload receipts pictures which are process in the server to extract the information in this order, using a yolo model for image section identification/extraction, Tesseract OCR for the text extraction and Chatgpt llm for formation and classification of the data. The system was evaluated by its capacity accurately extract the section of receipts to ensure accuracy of the following text extraction. Moreover, the key feature and component from the server and application are tested to ensure their well behavior. The analysis showed that the system considerately reduces the effort of logging of receipt data while marking a significant improvement over traditional manual extraction method.

# 2 Introduction

## 2.1 Problem statement

In the context either company or personal accounting which “plays a significant role in the effective management process” (Alabdullah, 2019), a major inefficiency lies in the manual process of receipt logging. This task, vital for recording diverse expenses, is traditionally done by hand, a practice than not only result in substantial paper waste but also consumes considerable human resources. This inefficient task heavily time consuming with also a high likelihood for large amount of logging, show furthermore an environmental impact. The production and disposal of paper receipts contribute to the growing ecological footprint of businesses. There is a pressing need to address both the human resource and environmental cost associated with this outdated practice. The core of the challenge in this project is: how can we facilitate the receipt logging process to be more time-efficient, resource-effective and environmentally friendly?

## 2.2 Background

After observing the resource consuming task of receipt logging through a family related company of digitalizing and logging receipt by hand and one by one for hours, has showed the centrality and difficulty of this task in company accounting. It is mandatory task for company but also wanted by a lot of individuals wanting to keeping track of their expense and manage budget. Existing solution exist such has “Recipator AI” (AI, no date) or “Veryfi” (*Transform Documents into Actionable Data in Seconds using Veryfi OCR API*, no date) but are limited in feature by paywalls and even through it are not personalize to the Mauritian market and adapted to the variety of receipt while missing for free resources of friendly user-interface. A digitalized automated system promises not only to free up valuable human resources but also to reduce paper usage, align with global ecological sustainability’s. The introduction of an adapted, state-of-the-art system for managing and extracting receipts information’s would not just streamline accounting practices but would also impact of paper receipt production and storage. The project is positioned at intersection of technological innovation and ecological work, aiming to deliver a solution that can be leveraged across diverse business and personal landscape in their accounting.

## 2.3 Aims

The project “Smart Receipt Management and Extraction” system is designed to streamline and simplify the repetitive and time-consuming task of handling, organizing and informatize receipts, whether physical or digital. Therefore, the primary purpose of this project is to serve a comprehensive receipt extraction and management system, offering the user a set of features allowing for a user-friendly user-interface experience.

## 2.4 Objectives

The project designed can be expanded into objectives which will guide the different research and development for the project to be well executed and professional. These objectives can be summarized into three components.

The ability to **extract receipts** is the major element of the project, and the application should be able to extract the key information from a receipt’s image. The extraction should be available for multiple type of receipts, with precision for it to be usable for analysis and storage.

The second objective is the **storing capacity** ability in the project. The storing system should be flexible and accessible for the different components to use the stored data. The storing will encode the receipt extraction process data, properly compose the raw image, and mediate data processing and the extracted data. It will also contain useful information for the system such as the user information.

The last objective is to **provide analytics** to the user. The system should use the extracted data of the receipts to generate various graphs and charts. The purpose is to be able to provide insight to the user about their purchase history and guide them in the potential management of a budget.

# 3 Literature review

## 3.1 Receipt Extractor

### 3.1.1 Paper 1 - Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms.

This paper from Joel Odd and Emil Theologou (2018) is a study that “investigated if it was feasible to use machine learning tools on OCR [Optical Character Recognition] extracted text data to classify receipts and extract specific data points”. This process firstly extracts the receipt data through different Optical Character Recognition (OCR), then classifies through a model into different categories. They have tested different technologies for the OCR extraction and the model prediction, all listed below in *Table 1*.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Technologies** | **Pros** | **Cons** |
| **Optical Character Recognition (OCR)** | Azure Computer Vision API | * Good accuracy in text recognition. | * The security is uncontrollable due to third party dependance. * May require significant data preprocessing. |
| Google Cloud Vision API | * Good performance and time response. | * The security is uncontrollable due to third party dependance. * Managing variations in receipt formats can be complex |
| Tesseract OCR | * Deployable locally * Not dependent of third party. * Strong community | * Varying receipt formats may affect accuracy. * Accuracy depends on image quality. * Slow process for large amount of data. |
| **Machine Learning Models** | Linear Support Vector Classification (LinearSVC) | * 94% accuracy achieved. * Efficient with large text dataset | * Not has efficient for non-linear data relationship. * Require optimal parameters tuning, otherwise reduction in performance. |
| Multilayer Perceptron Classifier (MLPClassifier) | * Can capture complex relationship (non-linear). * Flexibility with multiple parameters. | * May be computationally intensive as the model become complex. * Risk of overfitting if the parameters are not well set. |
| Naive Bayes Classifier | * Easy implementation to handle large dataset. * May be use has baseline for classification problem. | * Can make naïve assumption which will lead to a reduction in accuracy for complex data. * Training dataset balance strongly influence the accuracy. |

Table 1 - Advantage and Limitation of technology used in "Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms" written by Odd and Theologou

From the OCR, three principal technologies are tested “Azure Computer Vision API” provided by Microsoft, “Google Drive REST API” provide by Google, and “Tesseract OCR”. The “Azure Computer Vision API” such has “Google Drive REST API” are efficient and powerful OCR tools but are third party dependent and all their processing power are deported in their own servers. This deported strategy allows powerful OCR but creates delay using API to upload and download the data and create a dependence to their services and could be costly. On the other hand, “Tesseract OCR” is an open-source OCR which can be deployed locally and, therefore, have a quicker response time compared to “API” OCR. In the context of our project, the user will have the ability to correct any error from the extraction, therefore a quick response time is essential for a streamline user-experience and is preferable over a slice reduction of extraction accuracy.

After extracting the text from the receipt, the output is categorized through models. Different models were used, such as the Linear Support Vector Classification, the Multi-layer Perceptron classifier, and the Naive Bayes Classifier, which all have their individual advantages and disadvantages, which are shown in *Table 1*. Their model is based on a text extract before using the model strategy, and therefore, is not used for image recognition.

### 3.1.2 Paper 2 - Information Extraction from Scanned Invoices using Machine Learning, OCR, and Spatial Feature Mapping Techniques

The second research paper “Information Extraction from Scanned Invoices using Machine Learning, OCR and Spatial Feature Mapping Techniques” is a project focusing one extracting information from scanned invoice using different technologies for different step (Darsha, 2023). In the context of our receipt extractor project, the focus will be on the receipt detection/classification and the text extraction step only.

For the Optical Character Recognition, like in the first paper in the section *Paper 1 - Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms.*, similar technologies were used, such as Tesseract OCR and Google Cloud Vision API. Since we have already discussed the advantages and disadvantages of these technologies, there is no need to re-analyze them. There it shows that these technologies are the state of the art in their domain and prove their efficiency.

As for the text detection and classification, it uses the YOLO (You Only Look Once) model, which is a state-of-the-art object detection application provided by Ultralitics in python, that is shown in *Table 2*. The YOLO model is well known for its rapid speed in prediction, while having a low background mistake, which is an important characteristic for a receipt extraction application wherein the picture would be taken quickly from a mobile phone. However, it also has its own limitations such as the trade-off between speed and accuracy, which could cause problems in case of too low accuracy. Also, the processing could be resource-intensive for training and prediction.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Technologies** | **Pros** | **Cons** |
| **Object detection model** | YOLO version 5 | * Good accuracy in detecting and classifying multiple objects. * Fast inference time and global speed. * Reduction of background error by processing all picture | * Require extensive computation power for many classes. * Potential trade-off between speed and accuracy. * Less densely pack data may reduce accuracy. |
| **Optical Character Recognition (OCR)** | Tesseract OCR | See *Table 1* | |
| Google Cloud Vision API | See *Table 1* | |
| Convolutional Neural Networks (CNNs) | * Good at extracting hierarchical feature from images. * Versatile for a wide range of image recognition. | * High computational resources for training. * Risk of overfitting training data. * Need a lot of tuning and optimization for optimal performance. |

Table 2 - Advantage and Limitation of technologies used in "Information Extraction from Scanned Invoices using Machine Learning, OCR and Spatial Feature Mapping Techniques" written by Darsha.

### 3.1.3 Paper 3 - Computer Vision for Document Image Analysis and Text Extraction

The third paper “Computer Vision for Document Image Analysis and Text Extraction” is a research article aiming to improve Optical Character Recognition (OCR) systems, particularly for image processing (Benchekroun, 2022). It explores multiple technologies, such has Convolutional Neural Network (CNN) + Long Short-Term Memory (LSTM) Network for feature extraction from images, Deep CNN + Transformer/seq2seq Network to handle sequential data, Generative Adversarial Networks (GANs) to generate synthetic training data, and Morphological Operations for preprocessing training data with different operation.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Technologies** | **Pros** | **Cons** |
| **Optical Character Recognition (OCR)** | Convolutional Neural Networks (CNN) + Long Short-Term Memory (LSTM) | * Effective for non-handwritten text. * Shows significant accuracy improvement with synthetic data. | * Are computationally intensive and may require large dataset for high accuracy. * Struggle with complex patterns in data |
| Deep CNN + Transformer/seq2seq Network | * Capable of parallel processing, offering faster execution. * Can extract complex features from images. | * Require significant computation resources. * Difficulty to optimize due to his complexity. |
| **Synthetic Data Generation** | Generative Adversarial Networks (GANs) | * Increase training data. * Help model learn feature that may not be present in real-world scenario. | * Generated data might not always represent real-world scenarios. * Can reduce accuracy in case of generated data not accurate to the scenario. |

Table 3- Advantage and Limitation of technologies used in "Computer Vision for Document Image Analysis and Text Extraction" written by Benchekrou.

The advantages and disadvantages of these technologies can be found in *Table 3*. Furthermore, the usage of GANs to create synthetic data is an interesting feature to improve the training dataset and, therefore, increases the accuracy if an OCR algorithm is developed from scratch, such as the different ones used in this third research paper. Developing an OCR from scratch without using an already made system, such has Tesseract or Google Cloud Vision API, could allow for a more precise extraction especially for Mauritian receipts, thereby it will increase the development time, but it will be limited by its training data, which in out context, is limited and, therefore, would not be recommended compared to the powerful pretrained system.

## 3.2 Mobile Application

### 3.2.1 Paper 4 - React Native vs Flutter, cross-platform mobile application frameworks.

The last research paper “React Native vs Flutter, cross-platform mobile application frameworks” focuses on comparing two frameworks to develop cross-platform mobile application: React Native and Flutter (Wu, 2018). These two frameworks are prominent factors in mobile development, and their advantages and disadvantages are available in *Table 4*. React Native, which was developed by Facebook, is using JavaScript and React, which are famous programing languages. However, for complex development, it will require a specific programming language per platform. On the other hand, Flutter is developed by Google using Dart, which is not as widely adopted as JavaScript, but uses a single code base for both iOS and Android. However, this single code base creates a larger app size compared to React Native app.

In the context of the receipt extractor application, the emphasis will be on the execution speed and stability on the mobile app. Therefore, Flutter, as shown in *Table 4*, is fast with different features for development, such has the hot reload with the portability from Android to iOS without code modification.

|  |  |  |
| --- | --- | --- |
| **Programming Language** | **pros** | **Cons** |
| **React Native** | * Strong Community Support with big tech contribution. * Bring modern web techniques to mobile support. * Use JavaScript syntax extension for designing UI. * Access native hardware feature like camera and storage. * Encourage modularity and reusable of component. | * Have performance limitation compared to native app in complexes scenario using JavaScript thread or memory optimization. * Highly dependable on third-party libraries for navigation and file system operations which may affect consistency and the reliability. |
| **Flutter** | * Hight-Performance using his own rendering engine for view component offering a close performance to native application. * Using Dart Programming language which is efficient for memory management and garbage collection offering fast performance. * Providing customizable widget for development. * Hot-Reload feature for development. * Assuring a consistency across the different platform. | * Produce larger application size due to the different widget renderer that may affect the app size. * Relatively new community compared to react native which may affect the resources available. |

Table 4 - Advantage and limitation of Flutter and React Native from "React Native vs Flutter, cross-platform mobile application frameworks" written by Wu.

# 4 Requirements specification

## 4.1 Receipt Section Detection

As referenced while analyzing *Paper 2 - Information Extraction from Scanned Invoices using Machine Learning, OCR, and Spatial Feature Mapping Techniques ,* it presented technologies of detection of the different sections of a receipt before applying the text extraction (Darsha, 2023). These strategies will be kept by detecting the key element of the receipt to increase the formatting and classification of the receipt data by developing an image recognition model to identify the precise areas representing some elements of the receipt, such has the total, items list, time, and shop information.

## 4.2 Text Extraction

A key feature of the receipt extraction will be the character recognition which will allow the digitalization of the data from the paper to the receipt management system. As per the inside from the work from Odd and Theologou (Odd and Theologou, 2018)(*Paper 1*), Benchekrou (Benchekroun, 2022)(*Paper 3*) and Darsha (Darsha, 2023)(*Paper 2*) showing Optical Character Recognition (OCR) technologies, such has Tesseract OCR and Google Cloud Vision API, along with advanced machine learning models. Presented in all three research papers, it shows the importance of the text extraction as a vital feature as it impacts the precision of the global system.

## 4.3 Format and Classify text extraction.

After extracting the raw data from the receipt, a formatting and classification showed in *Paper 1 - Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms* and *Paper 2 - Information Extraction from Scanned Invoices using Machine Learning, OCR, and Spatial Feature Mapping Techniques* underscore the effectiveness of the machine learning algorithm for text classification using models such has LinearSVC and MLPClassifier. This approach facilitates the classification of the receipt and the formatting of the output to enable the uniformities of the data structure for a possible database storage and analytics.

## 4.5 User Interface

Even though the papers criticize the receipt extraction, it showcases multiple technologies for the processing, but none of them provide user-interface for common users to employ these technologies, which without this will restrict the usage of the software to specialized set of users. Therefore, a User Interface (UI) is required to increase the spectrum of possible users. As showed in *Paper 4 - React Native vs Flutter, cross-platform mobile application frameworks ,* comparing Flutter and React Native mobile developing framework (Wu, 2018). These frameworks demonstrated the portability of the UI development through multiple devise common use of the system.

# 5 Design

After discussing the existing work in *Literature review* and discussing the different requirements in *Requirements specification*, we can now discuss the design of the system application. At that stage of development, it can be separated into two distinct parts, which will be working together simultaneously. The server side, which will manage all the computationally exhausting processing, such as the “receipt extraction feature” which will reduce the execution time and provide a better user experience while communicating with the database. Also, everything will be built in a docker container to offer an easy deployment on any device. And on the other hand, the mobile application will be used as a gateway for the user to operate the system and communicate with the server. A Clearer view of the system and its intern interaction are showed in *Figure 1 - System Design Diagram* and *Figure 2 - System Class Diagram*. In the next following sections, a list of the features will be integrated.

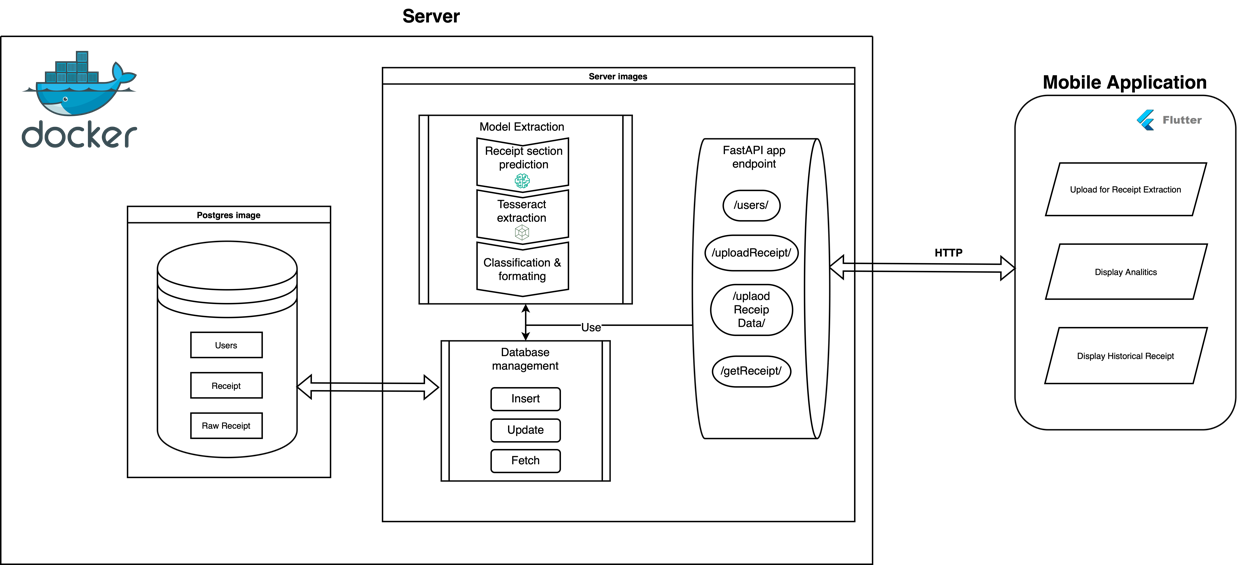


Figure 1 - System Design Diagram

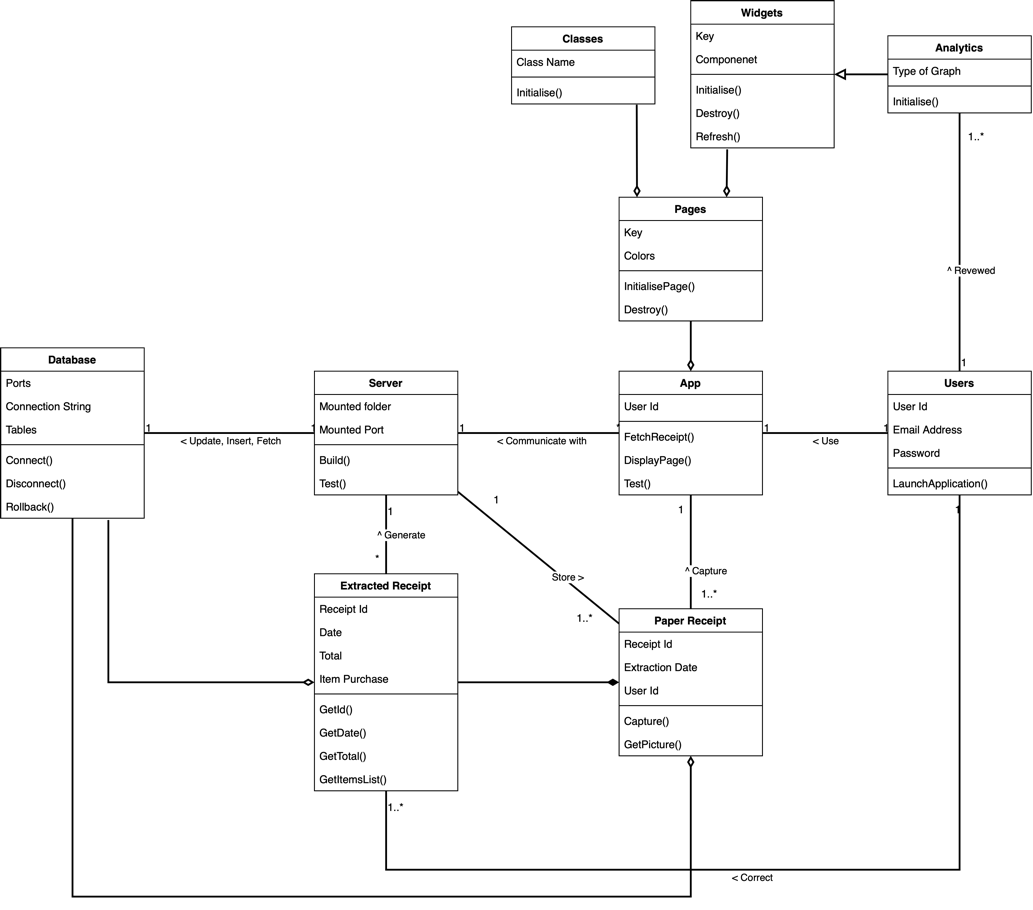


Figure 2 - System Class Diagram

## 5.1 Server

### 5.1.1 Receipt extraction feature.

As previously mentioned, the receipt extraction is the key feature of the system which has global purpose to extract from a receipt image into a specific format while predicting the type of receipt. The fields which will be extracted are the shop information, the list of items, the total, and the date of the transaction, which are to be formatted into a json format to be stored in a database. The structure of this feature is composed of three major parts; the receipt section image recognition model, followed by the text extraction of this section, and finally, the formatting and classification of the receipt extracted data.

The receipt section detection model is the first step of the receipt extraction feature. The model will compose of a YOLO v8 model to predict the different receipt sections, which are explained in the preceding paragraph trained by Mauritian receipt to better predict the section of the receipt. To describe in more detail, the prediction will take input as a picture of 640 per 640 pixel and return for each class (chop information, item list, …) their coordinate on the picture.

The second step of this feature is the extraction of the text display of these class extracts through OCR, for each detected class, a sub-image composed of coordinates of the class prediction, and the extracted text. As for the choice of the OCR, based on the different OCR technologies showed in the *Literature review*, the OCR used would be Tesseract OCR due to its high efficiency and capacity to be deployed locally without using a third-party service.

Finally, the classification and formatting of the of the output of the previous function output. This task will be performed by a Custom Multi-Layer Perceptron (MLP) to, firstly classify the receipt into categories (groceries, restaurant, cosmetics, electronics, etc.), then format all the information into a specific format (see *Figure 3 - Receipt Extraction Feature output format*) in json to be send back to the user. The user will then have the opportunity to correct the extraction result if wanted before being send back to the server to be stored into the database.



Figure 3 - Receipt Extraction Feature output format.

### 5.1.2 User Correction Feature.

While the data will be automatically extracted, the user will have the opportunity to “review” the extracted data, through the flutter application interface, to ensure the data is correct, which will be stored in the database. The corrected data will be set as “reviewed” and will be accessible by the analytics feature to ensure the accuracy of the data. The structure of the feature can be seen in the sequence diagram *Figure 5 - Receipt Extraction Feature Sequence Diagram*.

### 5.1.3 Server Communication

As for the communication between the server and the application, the HTTP (Hypertext Transfer Protocol) will be integrally handled by a FastAPI application host into the server. The primary function of the FastAPI app is to execute the requested server feature and relay back the result to the mobile app. Each endpoint will be related to a feature of the server and therefore, forming the server’s operation core.

### 5.1.4 Database

The server architecture will include a database system which will maintain user and receipt’s related data. Using Docker technology, the server will use a PostgreSQL database initializing alongside the server’s image. During the server’s image building, the PostgreSQL image will be initialized, and if in absence of pre-existing data, the database schema will be initialized through the execution of an SQL scripts. The usage of docker will ensure a seamless integration of the system.

The database schema is composed of three primary tables, outline as follows:

* Users table: A table for user related information.
* Raw Receipt Table: A table to store the unprocessed data before any data manipulation.
* Receipts Tables: A table which will contain all data related to the receipt extraction.

Each table use a specific purpose and all tables’ keys can be found in the database schema showed in *Figure 4 - Database Schema*.

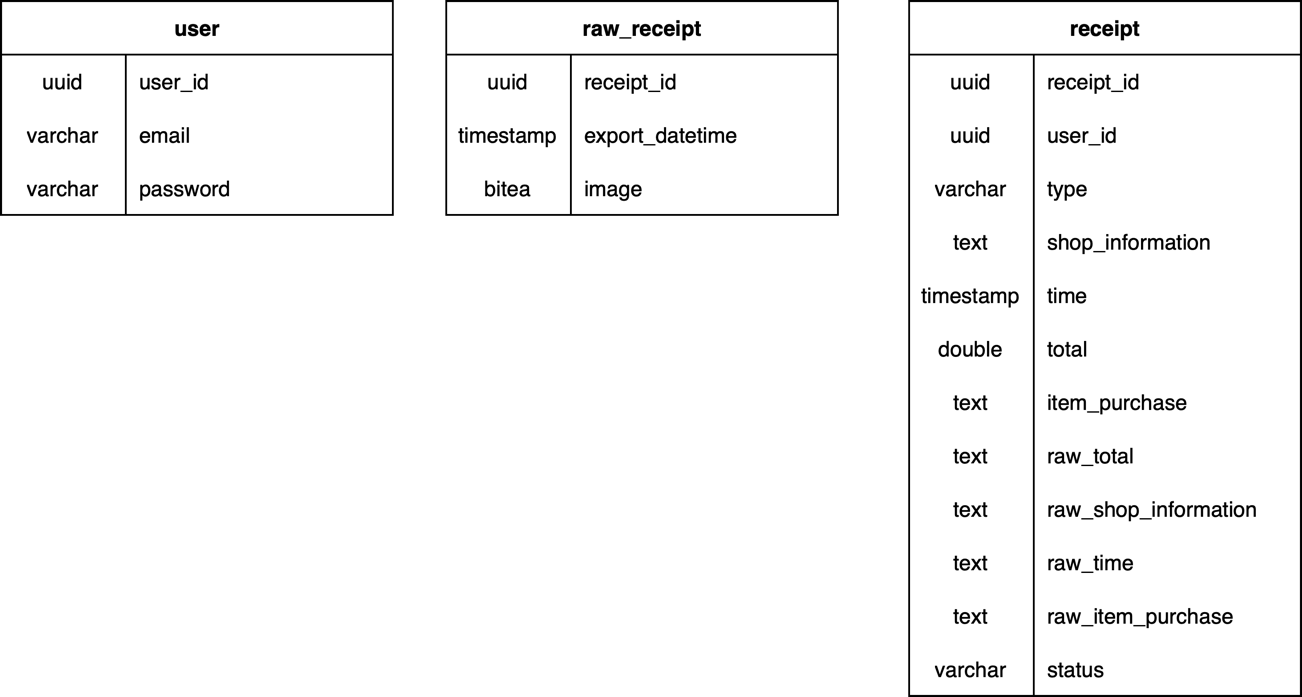


Figure 4 - Database Schema

## 5.2 Mobile Application

### 5.2.1 Receipt upload for receipt extraction.

On the user side, the key feature of the application is the upload of the receipt of the extraction of its data to be stored in the database. The first step is the loading of the receipt, either through the uploading from the phone gallery, or a live capture through the camera. It is then followed by the upload to the server through http post request, where the receipt extraction will be done. The result will be stored as “pending” status in the database before being sent back to the mobile application. With the result of the extraction, the user will have the option to review and modify the result to correct potential error before being sent back to the server to update the receipt data stored in the database and set is status as “reviewed”.

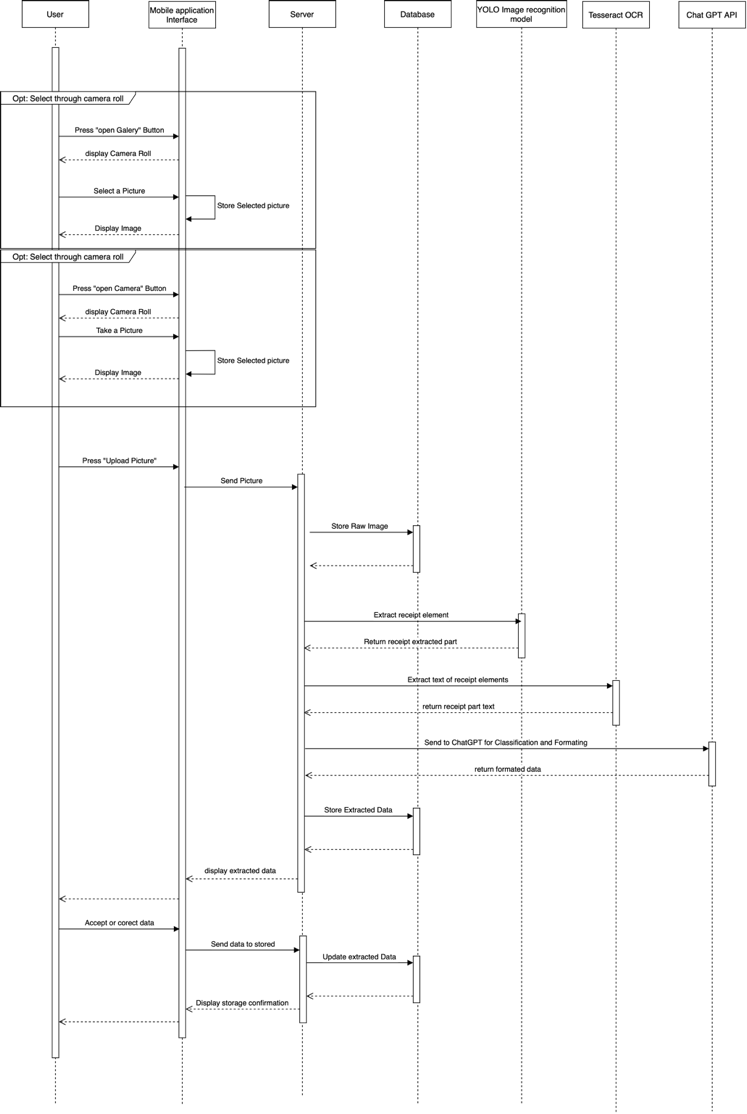


Figure 5 - Receipt Extraction Feature Sequence Diagram

### 5.2.2 List Receipt Historical

The application will allow the user to review their historical entries in a form of a list where key information about the receipt will be display. It will query the “reviewed” status receipt from the user before formatting the data for display.

### 5.2.3 Analytics

The application will offer the user the possibilities to show analytics through their data stored in the database. When this page has loaded, the current user’s data will be requested to the server, which will query it from the database back to the mobile application. The data will be processed directly from the application during the building of the widget to create different analytics for the user. The first graph provided will be a line graph showing the sum of purchase/s per day within a range of one month, with the capacity to display for older month/s. It will allow the user to keep track of the money spent during the month. A second graph provided will be generated as a pie graph that shows the sum of money spent per receipt categories (groceries, restaurant, etc.) per month, also with the ability to see data of older month/s. It will allow the user to view the amount for each category and allow it to manage a potential budget.

# 6 Implementation

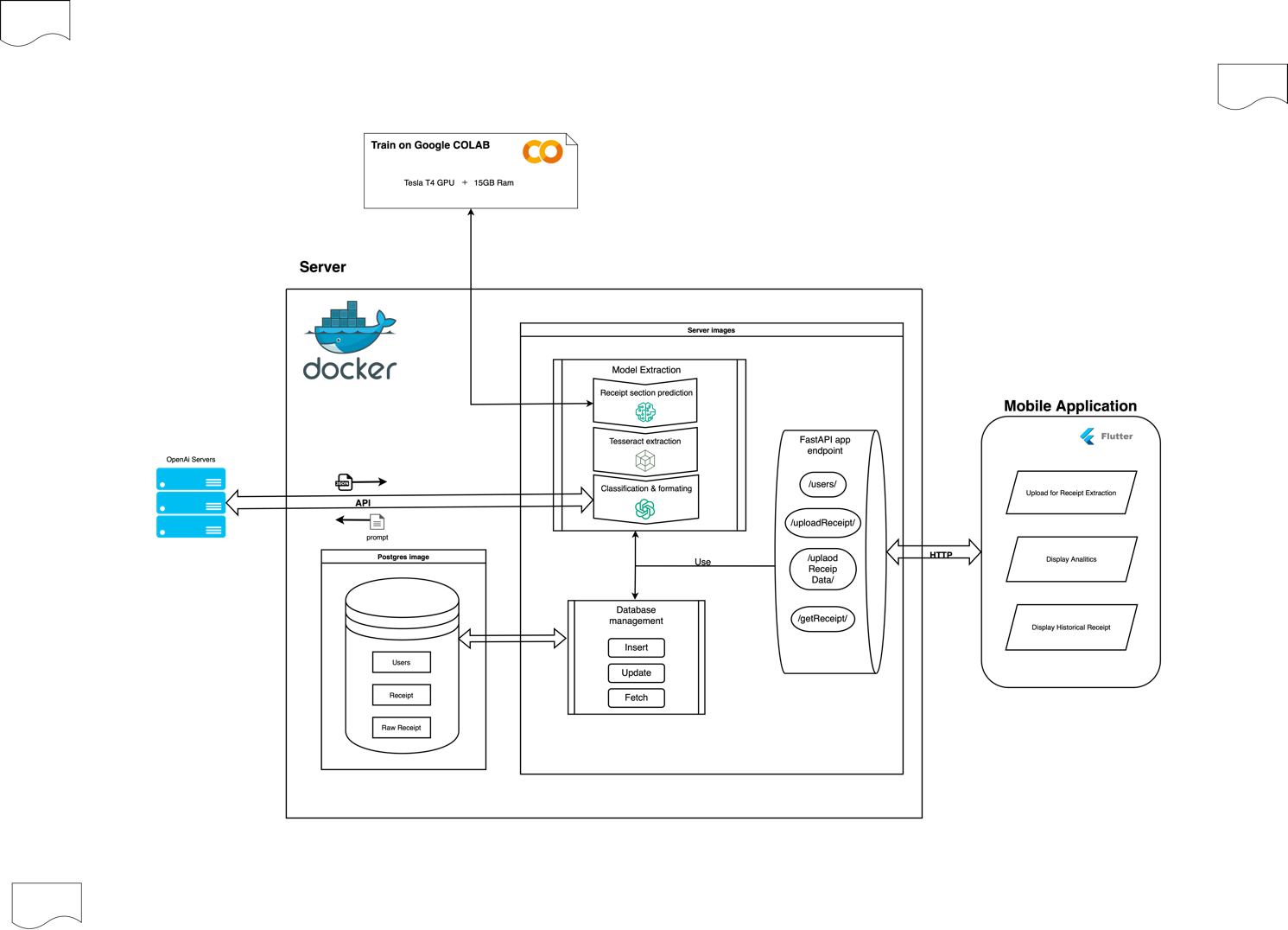


Figure 6 - System Implementation Schema

## 6.1 Server

During the development of the server, multiple technologies where used, alongside multiple developmental tools and techniques. This section is going to focus on the key component of the server and the modification needed to be done compared to the original design. In a first section, the integration of docker in the server will be discussed, followed by the usage of Makefile. And lastly, provide details about the Receipt Section Detection Model and the modification needed to be done for the classification/formatting section of the server. The core code of the server, the FastAPI endpoint can be seen in *Figure 7*.

|  |
| --- |
| # Endpoint to register a new user  @app.post("/register/")  async def create\_item(user: User):  try:  conn = get\_db\_connection(DB\_HOST, DB\_PORT, DB\_USER, DB\_PASSWORD, DB\_NAME)  cursor = conn.cursor(cursor\_factory=RealDictCursor)  user\_id = uuid.uuid4()  print("executing query")  insert\_query = "INSERT INTO \"users\" (user\_id, email, password) VALUES (%s, %s, %s)"  data\_to\_insert = (str(user\_id), user.email, user.password)  # Execute the query  cursor.execute(insert\_query, data\_to\_insert)    # return\_data = cursor.fetchall()  print(f"return\_data: {user\_id}")  conn.commit()  cursor.close()  conn.close()  return {"user\_id": user\_id}  except Exception as e:  conn.rollback()  print(f"Exception: {e}")  raise HTTPException(status\_code=500, detail=str(e))  finally:  cursor.close()  conn.close()  # Endpoint to get user details  @app.post("/users/", status\_code=200)  async def get\_user(kwargs: dict, response: Response):  try:  print(f"kwargs: {kwargs}")  conn = get\_db\_connection(DB\_HOST, DB\_PORT, DB\_USER, DB\_PASSWORD, DB\_NAME)  cursor = conn.cursor(cursor\_factory=RealDictCursor)  email = kwargs.get("email")  print(f"fetching for {email}")  select\_query = "SELECT \* FROM \"users\" WHERE email = %s"  data\_to\_select = (email,)  # Execute the query  cursor.execute(select\_query, data\_to\_select)  print("executed query")    return\_data = cursor.fetchall()  print(f"return\_data: {return\_data}")  if return\_data:  return return\_data[0]  else:  response.status\_code = status.HTTP\_204\_NO\_CONTENT  return {"detail": "User not found"}  except Exception as e:  conn.rollback()  print(f"Exception: {e}")  raise HTTPException(status\_code=500, detail=str(e))  finally:  cursor.close()  conn.close()  # Endpoint to upload a picture and extract the receipt data  @app.post("/uploadPicture/")  async def upload\_image(user\_id: str = Form(...), file: UploadFile = File(...)):  try:  contents = await file.read()  file\_location = os.path.join(UPLOAD\_PICTURE\_PATH, file.filename)  receipt\_id = str(uuid.uuid4())  print("User ID:", user\_id)    # Resize the image to 640x640  image = Image.open(BytesIO(contents))  resized\_image = image.resize((640,640))  # Save the resized image to the file location  with open(file\_location, "wb") as file\_object:  resized\_image.save(file\_object, format=image.format)    # Generate the raw receipt data  raw\_receipt\_data = {  "receipt\_id": receipt\_id,  "export\_datetime": datetime.datetime.now(),  "image": Binary(contents)  }  # Run your model and extract the receipt section  model\_results = extract\_receipt(model, file\_location)  # Format the extracted data  process\_results = generate\_receipt\_json(os.getenv("OPENAI\_API\_KEY"), model\_results)  clean\_data = clean\_receipt\_data(process\_results, receipt\_id=receipt\_id, user\_id=user\_id)  to\_upload = format\_extracted\_data\_for\_db\_upload(clean\_data, model\_results, receipt\_table\_column )  #insert into db  conn = get\_db\_connection(DB\_HOST, DB\_PORT, DB\_USER, DB\_PASSWORD, DB\_NAME)  cursor = conn.cursor(cursor\_factory=RealDictCursor)  cursor.execute(raw\_receipt\_insert\_query, raw\_receipt\_data)  cursor.execute(receipt\_insert\_query, to\_upload)  conn.commit()    # After saving the file, you can do additional processing if required  return {"image\_location": {file\_location},  "results": clean\_data}  except Exception as e:  conn.rollback()  print(f"Exception: {e}")  raise HTTPException(status\_code=500, detail=str(e))  finally:  cursor.close()  conn.close()  # Endpoint to upload receipt data and update the database  @app.post("/uploadReceiptData/", status\_code=200)  async def upload\_receipt\_data(kwargs: dict, response: Response):  try:  print(f"kwargs: {kwargs}")  conn = get\_db\_connection(DB\_HOST, DB\_PORT, DB\_USER, DB\_PASSWORD, DB\_NAME)  cursor = conn.cursor(cursor\_factory=RealDictCursor)  user\_id = kwargs.get("user\_id")  receipt\_data = kwargs.get("results")  receipt\_data["status"] = "reviewed"  receipt\_data["item\_purchase"] = str(receipt\_data["item\_purchase"]).replace("'", "\"")  for key, value in receipt\_data.items():  if value == "":  print(f"filling {value} with None")  receipt\_data[key] = None  # Check if it's a new receipt or an update  if kwargs.get("new\_receipt"):  print("Inserting new receipt data")  cursor.execute(new\_receipt\_insert\_query, receipt\_data)  else:  print("Updating receipt data")  cursor.execute(receipt\_update\_query, receipt\_data)    conn.commit()  cursor.close()  conn.close()  return {"status": "success"}  except Exception as e:  conn.rollback()  print(f"Exception: {e}")  response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR  return {"status": "failed", "message": str(e)}  finally:  cursor.close()  conn.close()  # Endpoint to get receipt data  @app.get("/getReceipt/")  async def get\_receipt\_data(user\_id: str = None):  try:  conn = get\_db\_connection(DB\_HOST, DB\_PORT, DB\_USER, DB\_PASSWORD, DB\_NAME)  cursor = conn.cursor(cursor\_factory=RealDictCursor)  select\_query = f"SELECT receipt\_id, type, shop\_information, time, total, item\_purchase FROM receipt WHERE status = 'reviewed' AND user\_id = '{user\_id}'::UUID ORDER BY time DESC"  cursor.execute(select\_query)  return\_data = cursor.fetchall()  print(f"Requesting data for {user\_id}")  cursor.close()  conn.close()  return return\_data  except Exception as e:  conn.rollback()  print(f"Exception: {e}")  raise HTTPException(status\_code=500, detail=str(e))  finally:  cursor.close()  conn.close() |

Figure 7 - Server FastAPI Code

### 6.1.1 Docker Usage

Since the server is the backbone of the whole system, the capacity to be easily deployed into the server with public IP to allow a connection through internet is important for more attainable access, and possibly reduce restriction while using the application. For developing purposes, the server is hosted locally using the hosting machine private IP to make the http request connect from the mobile application to the server with the objective in the future to be deployed into server with a public IP. The solution for an easy migration in the future is the usage containers technologies, and, in this case, Docker. Two images were built from the docker-compose.yml file. Firstly, the app image building coupled with the Dockerfile file sets the python environment of the app to Python3.9 and installs the different dependency and python packages required for the different processes for the server to run on. In addition, the whole server has been packaged and installed by the Dockerfile, and finally, at the end of the initialization of the app image, it starts the FastAPI application. The second image which was built is a Postgres image that provides a PostgreSQL database which the server will use to store data, is mounted onto a ‘/data’ folder. If this folder doesn’t exist, the initialization will build the database, its schema, and mock data by running the SQL file placed in ‘/postgres\_init’ folder. All these actions allow the server to be quickly and easy deployed into any kind of computer or server powerful enough as long as the support have docker.

### 6.1.2 Makefile

To facilitate the usage of the server and reduce the complexity of command to use the server, a Makefile is integrated with a set of rules useful for the user and uses the server and some specific case of the mobile application, the code of the makefile can be seen in *Figure 8*. There are 6 rules that can be use at the root of the project:

**‘make server-build’**: This rule is a simple set of commands which will go to the server code root and build the images of the server without starting it. It is mostly useful for developing purposes.

**‘make server-start’**: This set of commands does a bit more than just starting the server’s and database’s images. Due to the local usage of the server within local network, it firstly run bash script which get the current host machine IP and update a .env file which is used by the mobile application when building the executable (apk) to construct the host for the http requests. After updating the IP in the .env file, the rule gets into the server’s code root to start the server containers and FastAPI application.

**‘make server-stop’**: This rule, similarly to ‘make server-start’, has a purpose to change the status of the containers and stop them without the need to alter the IP.

**‘make rebuild-db’**: This set of commands is to be cautiously because it completely rebuilds the database by removing the current database. Firstly, it stops the containers from running, then removes the mounted database base folder ‘/data/’, and lastly, restarts the container, which will recreate a clean database base on the schema in the ‘/postgres\_init/’ folder.

**‘make server-test’**: This rule objective is to execute the server’s set of tests, but the complication is that for effectively testing the server’s features and endpoints as a local deployment, it needs to be run inside the docker directly. The Makefile rule then use the ‘exec’ command to connect to the container through this name and run the pytest set of tests to ensure the well execution of its features.

**‘make application-test’**: This set of commands, similar to the ‘make server-test’, aims to test the different features and widgets of the mobile application, but this time without the need to connect to a container. It just goes to the application code base and runs the flutter inbuild test command to run all of the test files in the ‘tests’ folder.

**‘make test’**: This rule has a simplified name as it integrates the execution of both testing the server and the mobile application. It chronologically executes the server’s set of tests, then the application’s set of tests by using the two previous testing rules. More details about the testing will be in *Testing*section.

|  |
| --- |
| .PHONY: build-server stop-server rebuild-db  # Path to your server directory  SERVER\_DIR = src/Server  APLLICATION\_DIR = src/MobileApp/receipt\_extraction\_project/receipt\_extractor  # Docker-compose service name for your database  DB\_SERVICE\_NAME = db  # Build the server  server-build:  @(cd $(SERVER\_DIR) && docker-compose build)  # Start the server  server-start:  @(cd $(SERVER\_DIR) && chmod +x update\_ip.sh && ./update\_ip.sh)  @(cd $(SERVER\_DIR) && docker-compose up)  # Stop the server  server-stop:  @(cd $(SERVER\_DIR) && docker-compose down)  # Rebuild the database completely  rebuild-db:  @echo "Stopping services..."  @(cd $(SERVER\_DIR) && docker-compose down)  @echo "Removing existing data directory..."  @rm -rf $(SERVER\_DIR)/data  @echo "Starting services..."  @(cd $(SERVER\_DIR) && docker-compose up)  server-test:  @docker exec -it server-app-1 pytest -v  application-test:  @echo "Running tests for the application..."  @(cd $(APLLICATION\_DIR) && flutter test lib)  test:  @echo "Running tests for the server..."  @make server-test  @echo "Running tests for the application..."  @make application-test  @echo "All tests passed!" |

Figure 8 - Makefile Code

### 6.1.3 Receipt Section Detection Model (YOLO)

As seen in the *Literature review*, there is multiple way to predict different element of an image, but the state of the art is Yolo provided by Ultralitics. To be more precise we are using YOLOv8 which is “the latest version of the acclaimed real-time object detection and image segmentation model”(Ultralytics, no date). Moreover, Ultralytics provide multiple size of the Yolo model (nano. Small, medium, large and extra-large) which we will use the biggest version yolov8x.pt for better performance.

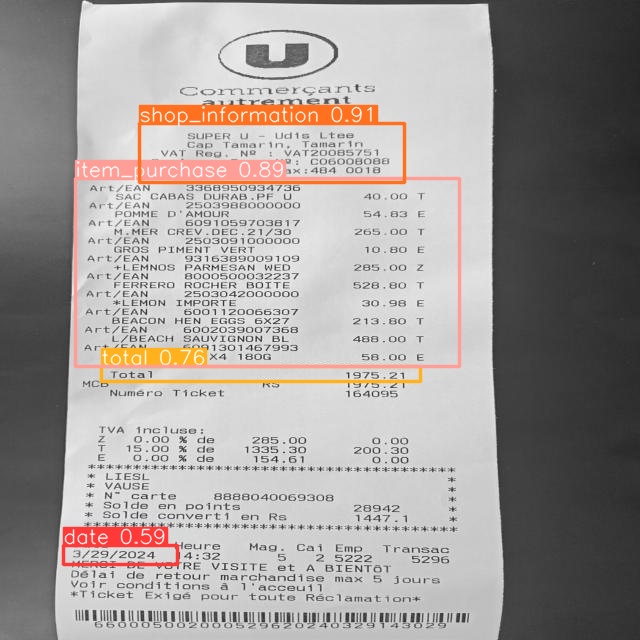
The model will take has input a reformat image of 640 per 640 pixels before applying the model to predict the coordinate of the different label. The set of labels represent the key section of the receipt containing the wanted information to be extracted afterward, the labels are the date, the shop information, total and the items purchase as showed in Figure 9. The usage of this text prediction serves as a cleaning of unhelpful in the context of the system such has the vat information or cashier information. This step reduces the amount of data which will be send to the classifier and formatter which will reduce potential error due to the high amount of parasite information. The python function to extract these sections can be seen in *Figure 10*.

Figure 9 - prediction model area with prediction percentage.

|  |
| --- |
| from ultralytics import YOLO  import cv2  from receipt\_server.model.functions import extract\_text  def extract\_receipt(model:YOLO, file\_path:str)->dict:  # Run your model  results = model(file\_path)  receipt\_data = {}  # Dictionary to store highest confidence detection for each class  highest\_conf\_detections = {}  for result in results:  for box in result.boxes:  class\_label = result.names[int(box.cls)]  conf = box.conf  # Check if this class\_label already exists and if this detection has higher confidence  if class\_label not in highest\_conf\_detections or highest\_conf\_detections[class\_label].conf < conf:  highest\_conf\_detections[class\_label] = box  # Now process only the detections with highest confidence for each class  for class\_label, box in highest\_conf\_detections.items():  image = cv2.imread(file\_path)  # Get box coordinates  x1 = int(box.xyxy[0][0])  y1 = int(box.xyxy[0][1])  x2 = int(box.xyxy[0][2])  y2 = int(box.xyxy[0][3])  crop\_img = image[y1:y2, x1:x2]  receipt\_data[class\_label] = extract\_text.extract\_text\_from\_image(crop\_img)  return receipt\_data |

Figure 10 - Extract Receipt Section Code

#### 6.1.3.1 Dataset

As for the dataset require for the training and testing, it was manually gathered from day to day living to fit the reality of receipt used to increase is efficiency in the most regular receipt provided. It is composed has for most of the receipt from Super U, London Supermarket, SmartBox, Intermart, Artisan Coffee, Central Electricity Board and Mauritius Telecom. From this database, the annotation was made using in a first time Roboflow web-application (*Roboflow*, no date) providing some good annotation and dataset increasing feature but mostly limited using free version, then were use Label-Studio which is an open source locally deployable labelling platform (*Open Source Data Labeling*, no date) which contrary to the free access of Roboflow doesn’t have an dataset export limit. The data is then exported into YOLO format making two folders, one for image and one for labeling sharing the same name. As for the dispatching of the dataset, Yolo work with a separation in 3, the training dataset which is used to train the model, the validate dataset used to evaluate the dataset during the training and use to avoid overfitting by tunning the model parameter, and finally the test dataset which is used to evaluate the final performance of the model after being trained with the train dataset which correspond has 66%(approximately) as showed in *Figure 11 - Dataset Distribution*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Receipt type | train | valid | test | total |
| London | 7 | 0 | 2 | 9 |
| Super U | 9 | 5 | 7 | 21 |
| Smart Box | 1 | 1 | 1 | 3 |
| Intermart | 5 | 2 | 2 | 9 |
| Artisan Coffee | 6 | 0 | 0 | 6 |
| Central Water Board | 21 | 3 | 3 | 27 |
| Mauritius Telecom | 15 | 1 | 1 | 17 |
| Old Brother | 0 | 0 | 1 | 1 |
| Winners | 0 | 0 | 1 | 1 |
| Emeralda Service Station | 0 | 0 | 1 | 1 |
| total | 64 | 12 | 19 | 95 |

Figure 11 - Dataset Distribution

#### 6.1.3.2 Training Process



Figure 12 - Model Training code used on Google Collab

The training of the model, despite how Ultralitic simplify it, it is still required for the larger model like the one we use, a lot of computational power, more than my personal laptop (Apple MacBook Air 15 equipped with a M2 chip) can process within a reasonable amount of time. During my test, the training took around 3 minutes per epoch using the laptop. A solution for this computational power is given by Google which provide temporally virtual machine with a powerful GPU using an Jupyter notebook show in *Figure 12*. To be more in details it provides an environment to execute python code using a Tesla T4 GPU with 15 GB of ram in addition of 12.7 GB of ram. Using this service, has allow a reduction of the training time to 3-4 seconds per epochs. The parameters from the yolo model get automatically tune by the training process using the validation dataset, the only parameters manually set is the epoch parameter which in the context of our system where set to 200 after multiple try and retry through multiple versions of the training dataset.

### 6.1.4 Classification/Formatting

As said before, in the receipt extraction feature, the extracted data of the receipt prediction from Tesseract OCR needs to be classify and formatted. Therefore, due to the quality of the extraction, which is messier than plan, which make the usage of a custom Multi-Layer Perceptron unrealistic for the complexity of task with the current setup of the system, amplified by the computational limitation of my personal laptop. This feature of classification and formatting being a key element in the receipt extraction feature, other solution needed to be tried to find a viable solution.

After multiple research for solution, language model (LM) sort out to be pretty efficient for classification purpose (Lenzmann, 2024). Therefore, in the optical of offering a cost-effective system to the user, a local deployment of a LM on the hosting machine could solve the current problem. A small officiant LM “Starling-LM-7B-alplha”(Jiao, 2023) using 7 billion parameters were tried to be deployed. But still due to the limitation of the host computation power, the response time during the test was way longer than it can be for a realistic usage (around 5 to 7 minutes per query) and therefore not be viable solution.

Since the problem being the computational power to host a LM, the solution is to search for a service which give access to LM while handling the heavy computation power to host it. The obvious and efficient solution is to use OpenAI API to access and use they Large Language Model (LLM) to act has classifier and formatter. The system therefore use “gpt-3.5-turbo” with the prompt show in Figure 13 to restrict the usage of the LLM into the classifying of the receipt data and to output only a json format has showed in *Figure 3 - Receipt Extraction Feature output format*. However, this solution comes at a cost price for the project, as OpenAI API is not available for free.



Figure 13 - Chatgpt classifying and formatting prompt.

## 6.2 Mobile Application

For the development of the user side, the mobile application is develop using Flutter framework from Google which is program in dart programing language offering a multiplatform which can be open on computers, Android and Apple IOS phone. The development was done on a Xiaomi 11T using Android 14 with Xiaomi Hyper OS overlay with some test on an emulation of an Apple iPhone 15 under IOS 17.4. The application code can be decomposed in 3 sections: classes, widgets and pages, which all together form the application.

### 6.2.1 Classes

The application class section is composed of two major classes. In the context of flutter, we call “class” a set of useful function which isn’t related to any visible element from the app. There are two feature which fit this description, the data services and receipt class.

The **data services** class has is name relate to, is managing the communication between the application and the server. All its functions are related to each endpoint of the server which are used by the application while formatting the data for the endpoint to accept it and send it through http protocol. The first function ‘fetchReceipts’ query from the server the users “reviewed” status data and format it to return a list of Receipt object (which will be discuss later). The second function is ‘sendPicture’ which format the image selected by the user to send it to the server for extraction, the result from the http is then return for further manipulation. The last function follows the logic of the receipt extraction, it’s the upsert of the reviewed and corrected back to the server to be upload into the database using the ‘sendReviewedData’ function from this class as showed in *Figure 14*.

|  |
| --- |
| class DataService {  Future<List<Receipt>> fetchReceipts(String user\_id) async {  var url = Uri.parse("http://${dotenv.env['CURRENT\_IP']}:8000/getReceipt?user\_id=$user\_id");  var response = await http.get(url);  if (response.statusCode == 200) {  List<dynamic> data = jsonDecode(response.body);  return data.map((json) => Receipt.fromJson(json)).toList();  } else {  throw Exception('Failed to load receipts : ${response.statusCode} - ${response.body} - ${response.request}');  }  }  Future<Response> sendPicture(XFile? \_imageFile, String user\_id) async {  if (\_imageFile == null) {  throw Exception('No image file selected');  }  final mimeTypeData = mime.lookupMimeType(\_imageFile!.path, headerBytes: [0xFF, 0xD8])?.split('/');  var request = http.MultipartRequest('POST', Uri.parse("http://${dotenv.env['CURRENT\_IP']}:8000/uploadPicture/"));  request.fields['user\_id'] = user\_id;  request.files.add(await http.MultipartFile.fromPath(  'file',  \_imageFile!.path,  contentType: MediaType(mimeTypeData![0], mimeTypeData[1]),  ));  var streamedResponse = await request.send();  var response = await http.Response.fromStream(streamedResponse);  return response;  }  Future<String> sendReviewedData(Map<String, dynamic> data) async {  final response = await http.post(  Uri.parse("http://${dotenv.env['CURRENT\_IP']}:8000/uploadReceiptData/"),  headers: <String, String>{  'Content-Type': 'application/json; charset=UTF-8',  },  body: jsonEncode(data),  );  if (response.statusCode == 200) {  return response.body;  } else {  return 'Failed to send data';  }  }  } |

Figure 14 - Data Services Class Code

As for the **Receipt** class, is a representation of the receipt data to be easily manipulated in the application. The main function of the class is ‘fromJson’ show in *Figure 15* which parse the response format from the http ‘getRecipt’ endpoint from dictionary of data to a list of Receipt object while parsing them into the right datatype. In case of error in the parsing log an error and fill the receipt with a default error value showing an error in the different filed. Other feature from the receipt object is to remap the data into a dictionary similar and the capacity to convert it into string with ‘toJson’ and ‘toString’ functions. These two features are mostly use for the edited of old data to fit in the editing feature and for the formatting for upload to the server. The code can be seen below in *Figure 16*.

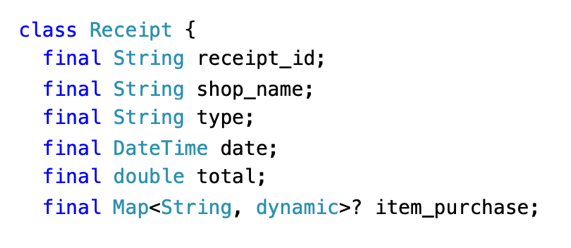


Figure 15 - Receipt class parameters.

|  |
| --- |
| class Receipt {  final String receipt\_id;  final String shop\_name;  final String type;  final DateTime date;  final double total;  final Map<String, dynamic>? item\_purchase;  Receipt({required this.receipt\_id,  required this.shop\_name,  required this.type,  required this.date,  required this.total,  this.item\_purchase,  });  static final \_logger = Logger();  factory Receipt.fromJson(Map<String, dynamic> json) {  try {  return Receipt(  receipt\_id: json['receipt\_id'] as String,  shop\_name: json['shop\_information'] as String,  type: json['type'] as String,  date: DateTime.parse(json['time']),  total: json['total'] as double,  item\_purchase: jsonDecode(json['item\_purchase'] as String),  );  } catch (e) {  \_logger.d('Error parsing receipt: $e');  return Receipt(  receipt\_id: 'Error parsing receipt',  shop\_name: 'Error parsing receipt',  type: 'Error parsing receipt',  date: DateTime.now(),  total: 0.0,  item\_purchase: null,  );  }  }  Map<String, dynamic> toJson() {  return {  'receipt\_id': receipt\_id,  'shop\_information': shop\_name,  'type': type,  'time': date.toIso8601String(),  'total': total,  'item\_purchase': item\_purchase,  };  }  String toString() {  return jsonEncode({"results":this.toJson()});  }  } |

Figure 16 - Receipt Class Code

### 

### 6.2.2 Widgets

Widget is a key feature in the Flutter framework, it gives reusability to created object leading to a global reduction in the code complexity while making it more readable by encapsulating them callable anywhere. In the context of the project, it was used to encapsulate complexes element which would have made the different pages codes too long and complexes and represent key element of the mobile app.

The **Navigation Bar** widget is the main app menu show in *Figure 17*, located at the bottom of every page of the application. It is used to navigate from a page to another. Technically speaking, each time a page is selected it overwrite the current page with the build function of the page selected and change the color of the button to display the current page display.

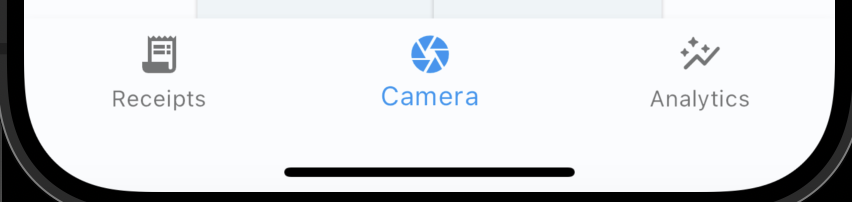


Figure 17 - Navigation bar.

|  |
| --- |
| class CustomBottomNavigationBar extends StatefulWidget {  final int selectedIndex;  CustomBottomNavigationBar({required this.selectedIndex});  @override  \_CustomBottomNavigationBarState createState() => \_CustomBottomNavigationBarState();  }  class \_CustomBottomNavigationBarState extends State<CustomBottomNavigationBar> {  late int \_selectedIndex;  @override  void initState() {  super.initState();  \_selectedIndex = widget.selectedIndex;  }  void \_onItemTapped(int index) {  if (\_selectedIndex != index) {  setState(() {  \_selectedIndex = index;  });  switch (\_selectedIndex) {  case 0:  Navigator.pushReplacement(  context,  MaterialPageRoute(builder: (context) => ReceiptsPage()),  );  break;  case 1:  Navigator.pushReplacement(  context,  MaterialPageRoute(builder: (context) => CameraPage()),  );  break;  case 2:  Navigator.pushReplacement(  context,  MaterialPageRoute(builder: (context) => GraphPage()),  );  break;  }  }  }  @override  Widget build(BuildContext context) {  return BottomNavigationBar(  key: const Key("BottomNavigationBar"),  items: const <BottomNavigationBarItem>[  BottomNavigationBarItem(  icon: Icon(IconData(0xe50d, fontFamily: 'MaterialIcons')),  label: 'Receipts',  ),  BottomNavigationBarItem(  icon: Icon(Icons.camera),  label: 'Camera',  ),  BottomNavigationBarItem(  icon: Icon(IconData(0xf59b, fontFamily: 'MaterialIcons')),  label: 'Analytics',  ),  ],  currentIndex: \_selectedIndex,  selectedItemColor: Colors.blue,  onTap: \_onItemTapped,  );  }  } |

Figure 18 - Navigation Bar Widget Code

The **camera** Widget show in *Figure 19* is used by the application “camera” page and is one of the two option to select a picture for the receipt extraction. When used for the first time, it ask the user to authorize the access to the camera to fit the European legislation for “General data protection regulation” (*General data protection regulation (GDPR) | EUR-Lex*, no date). When the authorization is provided, the user will be able to capture a picture by pressing the ‘Capture” button with the option to activate the phone flashlight if needed. When a picture is capture, the popup will close while passing the image to the “camera” page.

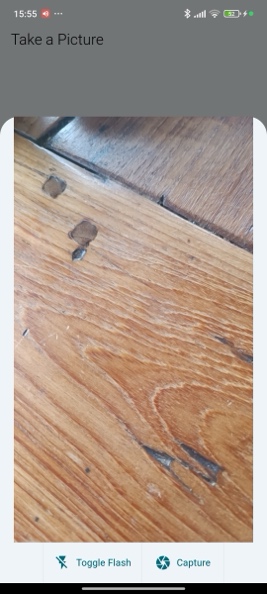


Figure 19 - Camera Widget

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:camera/camera.dart';  class CameraPopupWidget extends StatefulWidget {  final Function(XFile) onPictureTaken;  CameraPopupWidget({Key? key, required this.onPictureTaken}) : super(key: key);  @override  \_CameraPopupWidgetState createState() => \_CameraPopupWidgetState();  }  class \_CameraPopupWidgetState extends State<CameraPopupWidget> {  CameraController? \_controller;  List<CameraDescription>? \_cameras;  bool \_isFlashOn = false;  @override  void initState() {  super.initState();  \_initCamera();  }  Future<void> \_initCamera() async {  \_cameras = await availableCameras();  if (\_cameras!.isNotEmpty) {  setState(() {  \_controller = CameraController(\_cameras![0], ResolutionPreset.max);  \_controller!.initialize().then((\_) {  if (!mounted) return;  setState(() {});  });  });  }  }  void \_toggleFlash() {  if (\_controller != null) {  setState(() {  \_isFlashOn = !\_isFlashOn;  \_controller!.setFlashMode(  \_isFlashOn ? FlashMode.torch : FlashMode.off,  );  });  }  }  @override  void dispose() {  \_controller?.dispose();  super.dispose();  }  void \_takePicture() async {  if (\_controller!.value.isInitialized && !\_controller!.value.isTakingPicture) {  try {  await \_controller!.setFlashMode(\_isFlashOn ? FlashMode.always : FlashMode.off);  final file = await \_controller!.takePicture();  widget.onPictureTaken(file);  } catch (e) {  print(e);  }  }  }  @override  Widget build(BuildContext context) {  if (\_controller == null || !\_controller!.value.isInitialized) {  return Center(child: CircularProgressIndicator());  }  return Column(  children: [  Expanded(  child: CameraPreview(\_controller!),  ),  Row(  mainAxisAlignment: MainAxisAlignment.center,  children: [  ElevatedButton.icon(  style: ElevatedButton.styleFrom(  minimumSize: const Size(120, 60),  shape: const RoundedRectangleBorder(  borderRadius: BorderRadius.zero,  ),  padding: const EdgeInsets.all(16),  ),  onPressed: \_toggleFlash,  icon: Icon(\_isFlashOn ? Icons.flash\_on : Icons.flash\_off),  label: Text('Toggle Flash'),  ),  ElevatedButton.icon(  style: ElevatedButton.styleFrom(  minimumSize: const Size(120, 60),  shape: const RoundedRectangleBorder(  borderRadius: BorderRadius.zero,  ),  padding: const EdgeInsets.all(16),  ),  onPressed: \_takePicture,  icon: Icon(Icons.camera),  label: Text('Capture'),  ),  ],  ),  ],  );  }  } |

Figure 20 - Camera Widget Code

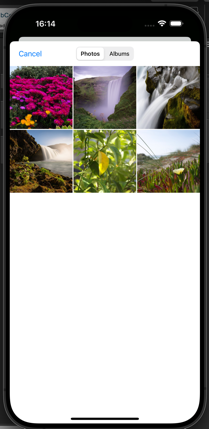
The **camera roll** Widget show in *Figure 21*, similarly, to the camera widget is the other way to select a picture to be send to the send to the database for extraction. To respect the same legislation, on the first use of the feature, an authorization prompt will pop up to give the application access to the phone gallery. The user will then be able to access his picture and albums to select the wanted picture which will be send back to the “camera’ page.

Figure 21 - Camera Roll Widget

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:image\_picker/image\_picker.dart';  class GalleryPopupWidget extends StatefulWidget {  final Function(XFile) onImageSelected;  GalleryPopupWidget({Key? key, required this.onImageSelected}) : super(key: key);  @override  \_GalleryPopupWidgetState createState() => \_GalleryPopupWidgetState();  }  class \_GalleryPopupWidgetState extends State<GalleryPopupWidget> {  final ImagePicker \_picker = ImagePicker();  void \_pickImageFromGallery() async {  try {  final pickedFile = await \_picker.pickImage(source: ImageSource.gallery);  if (pickedFile != null) {  widget.onImageSelected(pickedFile);  }  } catch (e) {  print('Failed to pick image: $e'); // Handle errors  }  }  @override  Widget build(BuildContext context) {  return Column(  mainAxisSize: MainAxisSize.min,  children: [  ListTile(  leading: Icon(Icons.photo\_library),  title: Text('Select from Gallery'),  onTap: () {  \_pickImageFromGallery();  Navigator.of(context).pop(); // Close the popup after selection  },  ),  ],  );  }  } |

Figure 22 - Camera Roll Widget Code

The **popup-receipt** show in *Figure 23* is a key widget reuse multiple time to display receipt data and edit it. It is used in multiple way and is a layer of security after extracting the receipt data, is used to edit the different part of receipt, newly extracted or add new receipt manually. The different fields have securities which display an error message in case of submission with an empty field. If needed the user can add or remove item through the trash icon button which remove the item or add an empty row for editing by pressing “add item” button while managing the different controllers for the text field. There is also a closing button to remove the popup and the submit button which use the data services class to upload the edited data to the server to be store into the database. The full code of the popup widget can be found in *Figure 24 - Popup Widget Code*.



Figure 23 - Popup Widget

|  |
| --- |
| import 'package:flutter/material.dart';  import 'dart:convert';  import 'package:uuid/uuid.dart';  import 'package:receipt\_extractor/globals.dart' as globals;  import 'package:receipt\_extractor/classes/data\_service\_class.dart';  class ReceiptPopup extends StatefulWidget {  final String buttonText;  final bool isNewReceipt;  final String popupText;  ReceiptPopup({Key? key, required this.buttonText, this.popupText = '', this.isNewReceipt = false}) : super(key: key);  @override  \_ReceiptPopupState createState() => \_ReceiptPopupState();  }  class \_ReceiptPopupState extends State<ReceiptPopup> {  late Map<String, dynamic> result;  late TextEditingController shopNameController;  late TextEditingController typeController;  late TextEditingController dateController;  late TextEditingController totalController;  late Map<String, TextEditingController> nameControllersDict;  late Map<String, TextEditingController> qtyControllersDict;  late Map<String, TextEditingController> priceControllersDict;  late Map<String, bool> validationErrors;  final \_formKey = GlobalKey<FormState>();  @override  void initState() {  super.initState();    if (widget.isNewReceipt) {  \_initNewReceipt();  } else {  \_initFromExistingReceipt();  }  }  void \_initNewReceipt() {  shopNameController = TextEditingController();  typeController = TextEditingController();  dateController = TextEditingController();  totalController = TextEditingController();    result = {"receipt\_id": Uuid().v1().toString()};  nameControllersDict = {};  qtyControllersDict = {};  priceControllersDict = {};  validationErrors = {  "Shop": false,  "Type": false,  "Date": false,  "Total": false,  };  }  void \_initFromExistingReceipt() {  Map<String, dynamic> response = json.decode(widget.popupText);  result = response['results'] ?? {};  shopNameController = TextEditingController(text: result["shop\_information"] != null ? result["shop\_information"].toString() : "");  typeController = TextEditingController(text: result["type"].toString());  dateController = TextEditingController(text: (result["time"] != null || result["time"] != "null") ? result["time"].toString() : "");  totalController = TextEditingController(text: result["total"] != null ? result["total"].toString() : "");  nameControllersDict = {};  qtyControllersDict = {};  priceControllersDict = {};  \_initItemFields();  validationErrors = {  "Shop": false,  "Type": false,  "Date": false,  "Total": false,  };  }  void \_initItemFields() {  result["item\_purchase"]?.forEach((key, value) {  nameControllersDict[key] = TextEditingController(text: key.toString());  qtyControllersDict[key] = TextEditingController(text: value['qty'] != null ? value['qty'].toString() : "");  priceControllersDict[key] = TextEditingController(text: value['price'] != null ? value['price'].toString() : "");  });  }  void \_addItem() {  String newItemKey = "Item ${nameControllersDict.length + 1}";  print("adding item $newItemKey");  setState(() {  nameControllersDict[newItemKey] = TextEditingController();  qtyControllersDict[newItemKey] = TextEditingController();  priceControllersDict[newItemKey] = TextEditingController();  });  }  List<Widget> \_buildItemFields(double \_fontSize) {  return nameControllersDict.keys.map((key) {  return \_buildItemRow(key, \_fontSize);  }).toList();  }  Widget buildBasicResponseField(TextEditingController textController, String category, double \_fontSize) {  return Column(  children: [  Container(  margin: const EdgeInsets.symmetric(vertical: 8),  child: Row(  children: [  Container(  height: 48,  alignment: Alignment.center,  padding: const EdgeInsets.symmetric(horizontal: 8),  child: Text(  '$category: ',  key: Key("${category} text"),  style: TextStyle(  fontSize: \_fontSize+5,  fontWeight: FontWeight.bold,  ),  ),  ),  const SizedBox(width: 8),  Expanded(  child: Container(  constraints: const BoxConstraints(minHeight: 48),  decoration: BoxDecoration(  border: Border.all(  color: (validationErrors[category] ?? false) ? Colors.red : Colors.grey,  width: 2  ),  borderRadius: BorderRadius.circular(5),  ),  child: TextFormField(  key: Key("${category} field ${validationErrors[category] ?? false ? 'error' : ''}"),  controller: textController,  decoration: InputDecoration(  border: InputBorder.none,  hintText: category,  contentPadding: const EdgeInsets.symmetric(vertical: 12, horizontal: 10),  ),  style: TextStyle(fontSize: \_fontSize),  validator: (value) {  bool isInvalid = value == null || value.isEmpty;  setState(() {  validationErrors[category] = isInvalid;  });  if (!isInvalid) {  return null;  }  return "This field is required";  },  ),  ),  ),  ],  ),  ),  const Divider(  color: Color.fromARGB(255, 84, 69, 50),  thickness: 1.5,  height: 15,  ),  ],  );  }    Widget \_buildItemRow(String key, double \_fontSize) {  return Padding(  padding: const EdgeInsets.only(bottom: 8.0),  child: Row(  mainAxisAlignment: MainAxisAlignment.spaceBetween,  children: [  Flexible(  flex: 4,  child: Padding(  padding: const EdgeInsets.only(right: 8.0),  child: TextField(  key: Key("$key Name Field"),  controller: nameControllersDict[key],  textAlign: TextAlign.center,  decoration: InputDecoration(  hintText: 'Name',  contentPadding: const EdgeInsets.symmetric(vertical: 10.0),  border: OutlineInputBorder(  borderRadius: BorderRadius.circular(10),  ),  ),  style: TextStyle(  fontSize: \_fontSize,  ),  maxLines: null,  ),  ),  ),  Flexible(  flex: 2,  child: Padding(  padding: const EdgeInsets.only(right: 8.0),  child: TextField(  key: Key("$key Qty Field"),  controller: qtyControllersDict[key],  textAlign: TextAlign.center,  decoration: InputDecoration(  hintText: 'QTY',  contentPadding: const EdgeInsets.symmetric(vertical: 10.0),  border: OutlineInputBorder(  borderRadius: BorderRadius.circular(10),  ),  ),  style: TextStyle(  fontSize: \_fontSize,  ),  ),  ),  ),  Flexible(  flex: 2,  child: TextField(  key: Key("$key Price Field"),  textAlign: TextAlign.center,  controller: priceControllersDict[key],  decoration: InputDecoration(  hintText: 'Price',  contentPadding: const EdgeInsets.symmetric(vertical: 10.0),  border: OutlineInputBorder(  borderRadius: BorderRadius.circular(10),  gapPadding: 2.0,  ),  ),  style: TextStyle(fontSize: \_fontSize,),  ),  ),  IconButton(  key: Key("$key delete button"),  icon: const Icon(Icons.delete),  onPressed: () {  setState(() {  nameControllersDict[key]?.dispose();  qtyControllersDict[key]?.dispose();  priceControllersDict[key]?.dispose();  nameControllersDict.remove(key);  qtyControllersDict.remove(key);  priceControllersDict.remove(key);  });  },  ),  ],  ),  );  }  @override  Widget build(BuildContext context) {  final double \_fontSize = 13;  return AlertDialog(  key: const Key('Receipt Popup'),  title: const Text('Popup'),  backgroundColor: const Color.fromARGB(255, 255, 255, 255),  content: SingleChildScrollView(  child: Form(  key: \_formKey,  child: Column(  mainAxisSize: MainAxisSize.min,  children: [  buildBasicResponseField(shopNameController, "Shop", \_fontSize),  buildBasicResponseField(typeController, "Type", \_fontSize),  buildBasicResponseField(dateController, "Date", \_fontSize),  buildBasicResponseField(totalController, "Total", \_fontSize),  ...\_buildItemFields(\_fontSize),  ElevatedButton(  key: const Key('Add Item Button'),  onPressed: \_addItem,  child: const Text('Add Item'),  ),  ],  ),  )  ),  actions: <Widget>[  TextButton(  child: const Text('Close'),  onPressed: () {  Navigator.of(context).pop();  },  ),  TextButton(  child: const Text('Submit'),  onPressed: () {  if (\_formKey.currentState!.validate()) {    Map<String, dynamic> updatedResult = {  "shop\_information": shopNameController.text,  "type": typeController.text,  "time": dateController.text,  "total": totalController.text,  "item\_purchase": {}  };  for (var entry in nameControllersDict.entries) {  updatedResult["item\_purchase"][entry.value.text] = {  "name": nameControllersDict[entry.key]?.text,  "qty": qtyControllersDict[entry.key]?.text,  "price": priceControllersDict[entry.key]?.text  };  }  updatedResult["user\_id"] = globals.user\_id;  updatedResult["receipt\_id"] = result["receipt\_id"];  String updatedJson = json.encode({  "results": updatedResult,  "new\_receipt": widget.isNewReceipt  });  print(updatedJson);  DataService().sendReviewedData(json.decode(updatedJson));  Navigator.of(context).pop();  }  },  ),  ],  );  }  }  void showPopup(BuildContext context, String buttonText, String popupText, bool isNewReceipt) {  showDialog(  context: context,  builder: (BuildContext context) {  return ReceiptPopup(buttonText: buttonText ,popupText: popupText, isNewReceipt: isNewReceipt);  },  );  } |

Figure 24 - Popup Widget Code

The different graphs display on the “graph” page are also encapsulate as widget as **line chart** and **pie chart** widgets. These widgets take before building the data previously fetch from the database and format them to format the corresponding chart to be display. Per example for the line chart, it filters to the corresponding month while summing the total value per day. From formatted data, the widget draws the chart for the corresponding month. The user also has the option to change the displayed month through a selection field allowing to draw older month as showed in *Figure 25 - Analytics Widgets States*.

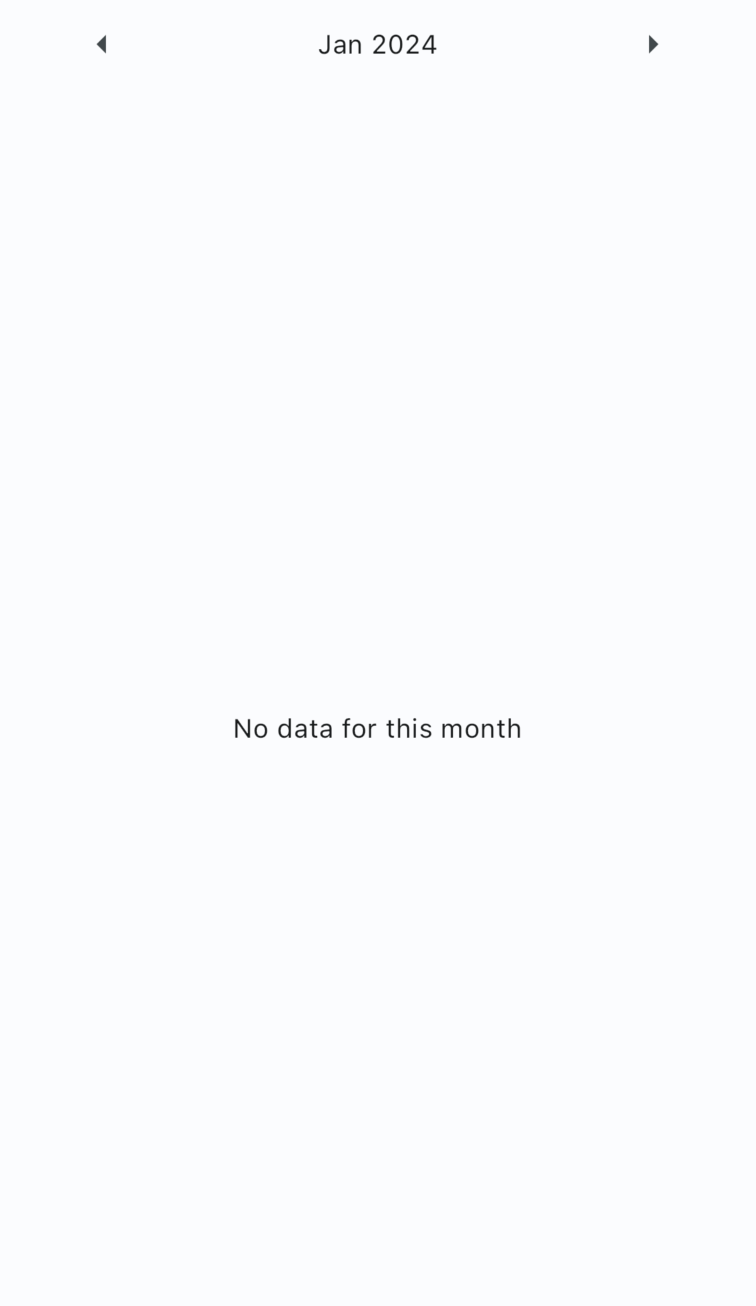
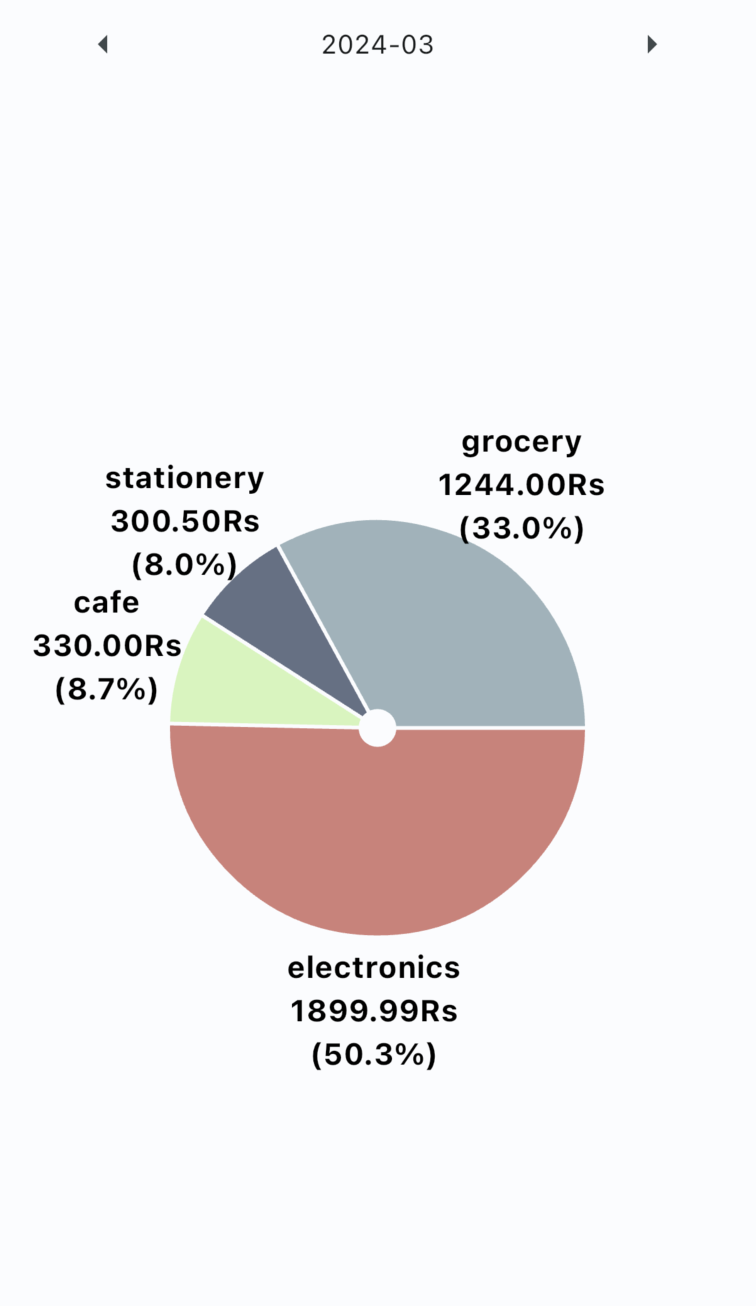
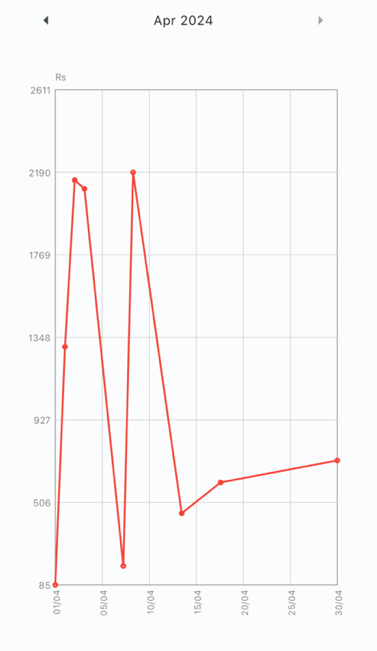


Figure 25 - Analytics Widgets States

### 6.2.3 Pages

Compared to classes and widgets which are tools to be used for the development of the application. On the other hand, pages use these tools to give a flowless and user-friendly experience to the application users. The application currently contains three pages all connected by the Navigation Bar widget.

The first page display by the application when launched, is the **camera page**. This page manages is use for the key feature of the system, the receipt extraction. As showed in Figure 26 - Camera Page States, it opens a blank page with two button each link to a widget. The first button is ‘Open Camera’ which when press trigger the camera widget to select take a picture. Next to it is the ‘Open Gallery’ which trigger the camera roll widget and allow the user to select an old picture. When a picture is selected from either the camera or the camera roll, it will be display at the center of the screen to ensure that it is the picture selected, and only when an image is display that the “extract” button appear. When press, it’ll use the data services class to upload the picture to the server for receipt extraction, during the extraction process in the server, a rolling animation will be display on the button as showed in third extract of *Figure 26*. When the server finishes the extraction, the response is then parse and display in the popup widget for user review of the extracted data to ensure his accuracy. After being edited, the user will either be able to cancel process by closing the popup or confirm the edited or not data to be send back to the database and be store has reviewed. The full code for the Camera page is available in *Appendix 1 - Camera Page code*.

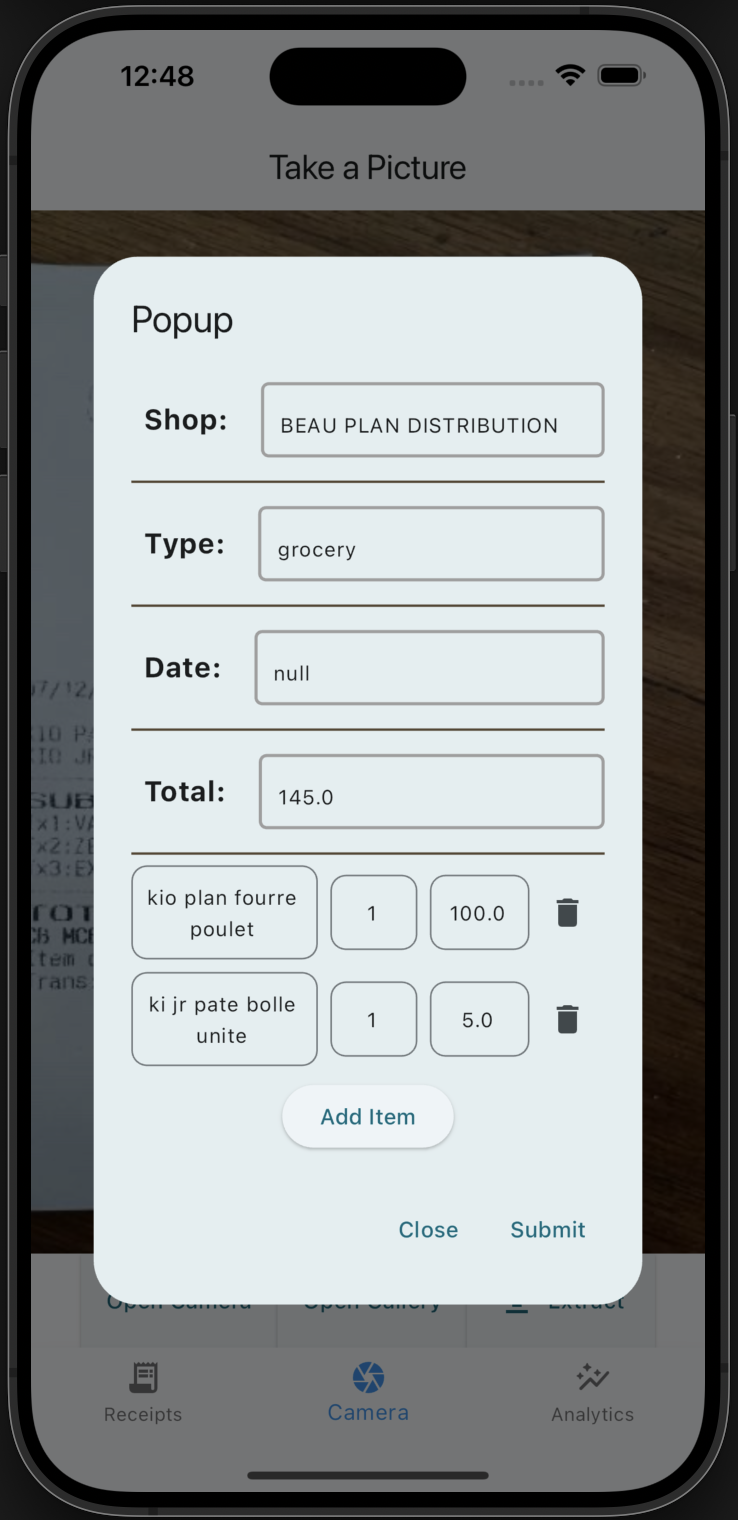
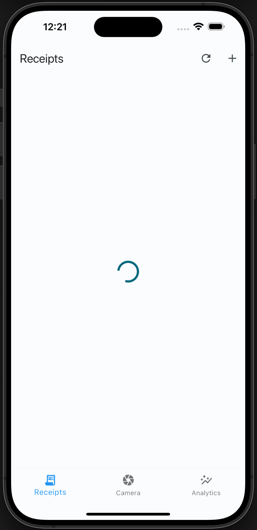
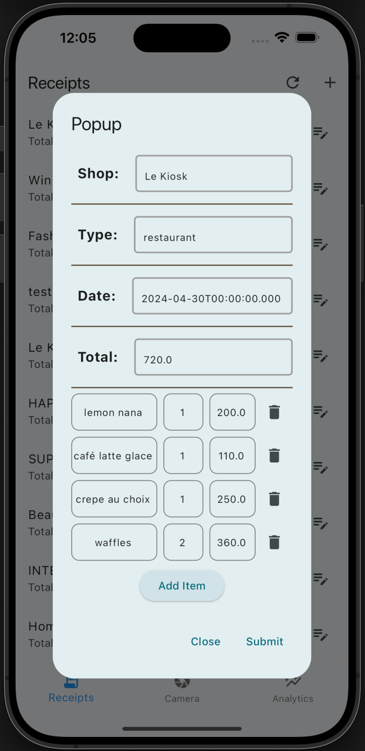
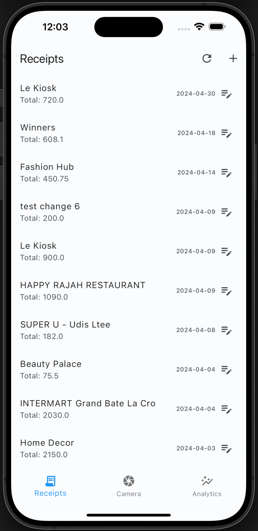


Figure 26 - Camera Page States

The second page accessible by the use is the **List Page**, this page has for purpose to display in form of a list all the user receipt from the more recent to the oldest. When the page load it will get the user’s “reviewed” data using the data services class while displaying a loading animation. If any error appends during the loading process due to connection error or any other potential error, an error message will be display. A specific message is also display if no data exist for the user. If the response contains receipt, it will be parse into a list of receipt using the Receipt class, from this list of receipt is generated the list which is display while showing the key information to the user. The list display show for each receipt the name of the shop, the total and the date of the receipt. In addition, a small edit button at the end of each row is displayed to give the use the opportunity to see in detail the receipt and the items that it contains or edit it before updating the data store in the database using the popup widget. As well as the list of receipt, there is two buttons at the top, one to reload the data from the database and redisplay it, and an add button to manually create an entry to be send to the database still using on the popup widget. The data manually entered by the user is automatously set as “reviewed” and will be used for analytics. The different states of the list page can be seen in *Figure 27* and the full code in *Appendix 3 - List Page Code*.

Figure 27 - List Pages States



The last page of the application is the **Graph Page**. This page gathers the different chart and analytics to be display on this page. During the loading of the page, the data services class gather the users “reviewed” data while a loading animation is display and appropriate message is display in case of loading error. At the top of the page is a tabBar menu which load the analytics widget by giving them access to the gathered data. The details about the different analytics are in *Widgets* section and the different states are showed in *Figure 28* and the full code of the graph page is in *Appendix 2 - Graph Page Code*.

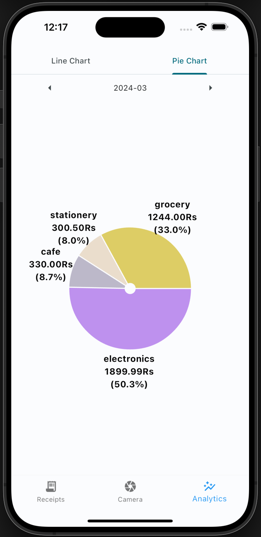
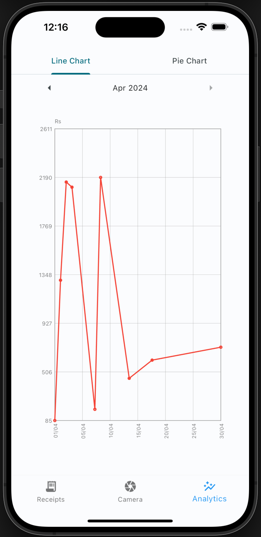
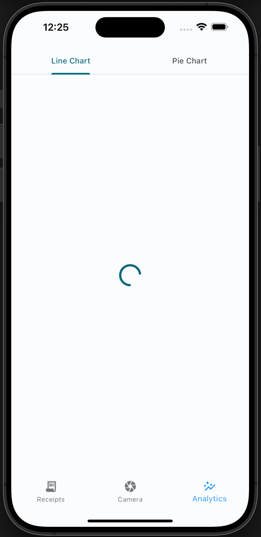


Figure 28 - Graph Page States

# 7 Testing

Testing is a key component of the development cycle of an application to ensure the well execution of all the features of the system and ensure the integrity of the whole system while developing future features. The project development is using the waterfall framework, which “separates the different parts of the project into phases specifying the necessary activities and steps” (*What Is the Waterfall Methodology? (Definition + Phases) | Built In*, no date) showed in *Figure 29 - Waterfall Framework Steps*. In the context of the project, the test can be separated into two parts; the mobile application side testing the different features, class, and widgets, and the server side wherein it tests the FastAPI features and tests the accuracy of the Receipt Section Detection Model.

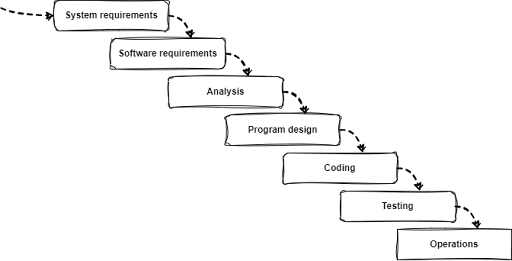


Figure 29 - Waterfall Framework Steps

## 7.1 Application

The testing of the application aims to ensure the skillful initialization and usage of the different classes, widgets, and pages. All the tests for the mobile application are based on the flutter base testing package, which separates the tests into groups concatenating multiple tests per feature. To perform those tests, a specific rule in the Makefile ‘make application-test’ is set at the root of the whole project, which carries out all the different tests present in the ‘lib/tests’ folder where the different tests are structured per feature. An example of the output log can be seen in *Figure 30 - Flutter testing result logs*.

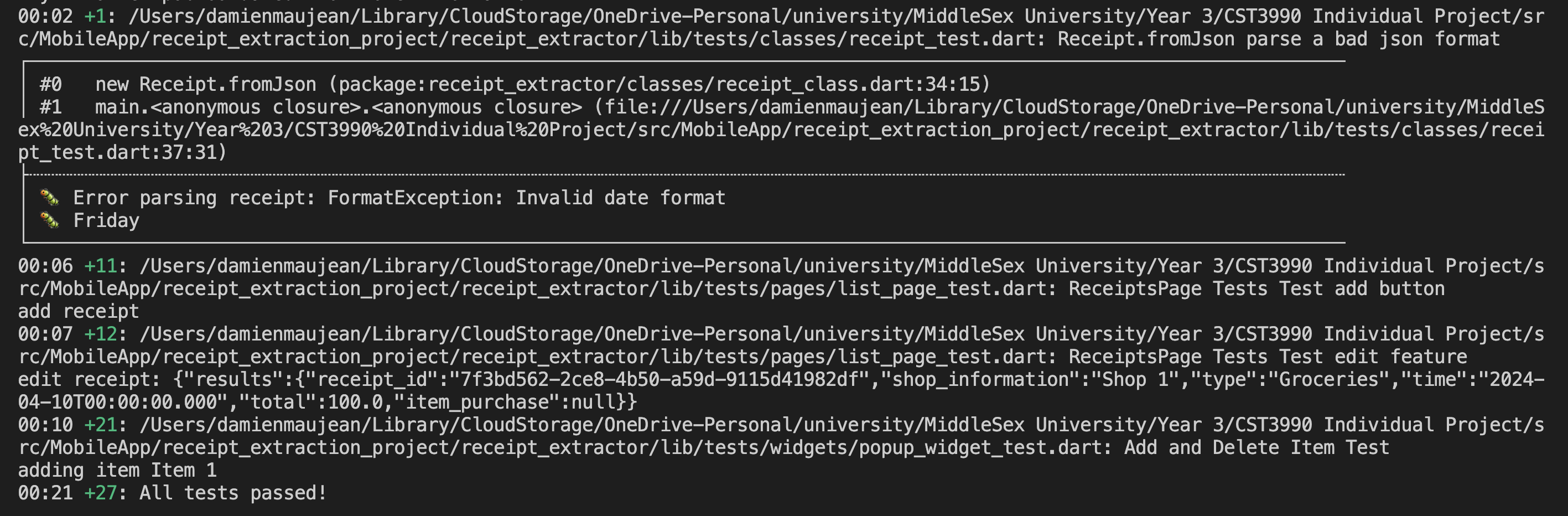


Figure 30 - Flutter testing result logs.

### 7.1.1 Classes

As referred in the implementation of *Classes* under *Mobile Application* section, there are two elements used as class object: the data services and the receipt classes. These tests ensure the well execution of the different features of these classes.

For the **Data Services Class**, the test focuses on the polished completion of the function communicating with the server. It firstly tests the retrieval of receipts from the server, then ensures that it can effectively send the picture to the server and receive quality response, and lastly test the upload of receipt data into the server.

As for the **Receipt Class**, the test focuses on the initialization of data using adequate inputting format for a successful initialization during first usage, then it ensures a satisfactory reaction of the initialization using wrong format which doesn’t correspond to the need format.

### 7.1.2 Widgets

The testing of the widget is different compared to the type of tests done for the classes. Along with the test of the different features, we need to test the acceptable display of the different elements which is composed of the widgets showed in *Widgets* section.

The testing of the **Navigation Bar Widget** focuses on the well-formulated initialization of the widget with the presence of all its key components. The second test then evaluates each button from the navigation bar independently and ensures that they load their designated pages correctly.

The **Pop-up Widget** tests similarly like the other widget, ensuring the well-formatted initialization and the different features of the widget. It ensures the structured placement of the different fields, and that the data is well displayed if initialized with data, then ensures the effective implementation of the “add item” and “delete item” button and finishing with the closing button.

As for the analytics, the same type of test is applied for the **Line Chart** and **Pie Chart** widgets. It firstly ensures the efficient production of the widget with data, then verifies the execution of the month changing and the message display if there is missing data for these months.

### 7.1.3 Pages

The last element from the mobile application which was tested is the pages, which encapsulates the widgets and classes to form the application. These are key elements which allow the user to be displayed and, therefore, must be thoroughly tested. Ensuring the test was created sufficiently, some modifications of the test were needed on the pages. The page should be able to load while overwriting the Data Services class to return a precise value during the initialization due to the testing environment. Therefore, the tests use personalized Data Services classes to initialize successfully.

For the **Camera Page**, its tests start with the initialization of the page as default without any picture selected and displays, which ensures the presence of all its button. Then the different buttons “Open Camera” and “Open Gallery” load the right widgets. To test the display the well display of the image on the page, some modifications were made on the loading of the page to initialize the page with an image which will be display.

The tests of the **List page** used their custom Data Services classes to test the initialization of the page and the generation of the receipt list firstly with some data and a visible list, then without data while displaying a no data message. Then they are tested the two buttons on the page: “add” and “edit” buttons. The add button ensures the initialization of an empty popup widget, and the edit button ensures the initialization of the same widget with the selected receipt data.

The final element tested is the **Graph Page**. It tests the initialization of the page with and without data using the same custom type Data Service class to manipulate the data loading during the initialization of the page to ensure the appropriate loading of the different menus and graph widget or missing data message. Then, another test consists of ensure well behavior of the graph menu display the right widget.

The final element tested is the **Graph Page**. It tests the initialization of the page with and without data using the same custom type Data Service class to manipulate the data loading during the initialization of the page to ensure the appropriate loading of the different menus and graph widget or missing data message. Then, another test consists of ensure well behavior of the graph menu display the right widget.

## 7.2 Server

The purpose of testing the server was to test the key features of the server and ensure their functioning execution. To do so, tests were made to evaluate the execution of the different endpoints and the function it is composed of. As for the Receipt Section Detection Model, the model was evaluated using different metrics.

### 7.2.1 Endpoint (FastAPI App)

The FastAPI application, which run in the server docker, is the core of the server running in a docker container, ss talked about in *Server* implementation section. The tests were created using python Pytest package which provided useful testing tools in python and allow it to run multiple sets using ‘pytest’ command. Due to the local deployment of the server without a fixed address, to test the different endpoints, it needs to be run inside the docker. Therefore, a rule from the Makefile (‘make server-test’) connect into the server container and run pytest command to execute the set of tests by connecting directly to the FastAPI app and execute multiple tests on the different endpoints.

The **‘/getReceipt/’** endpoint is tested to ensure the user receipt data from the database is gathered efficiently. It is tested with a valid and an invalid receipt id using the mock data, which is automatically input into the database at the database initialization.

For the **‘/uploadPicture/’** endpoint, the tests are important as the start of the key feature of the system. It firstly ensures that the picture sent is well stored when received, then the extraction process is executed and ensures the result format is proportionate. It ensures the result code and the presence of the prediction, its format, and the presence of all the field referenced in *Figure 3 - Receipt Extraction Feature output format*. The next endpoint that was tested is **‘/uploadReceiptData/’**, which is the logic following uploading a picture. Two main tests were executed for this endpoint. Firstly, it verifies the input result with a valid format, and then tries with an invalid format input expecting a 500-error code and specific error message. The log of the output of the test is seen in *Figure 31 - Server test result logs*.

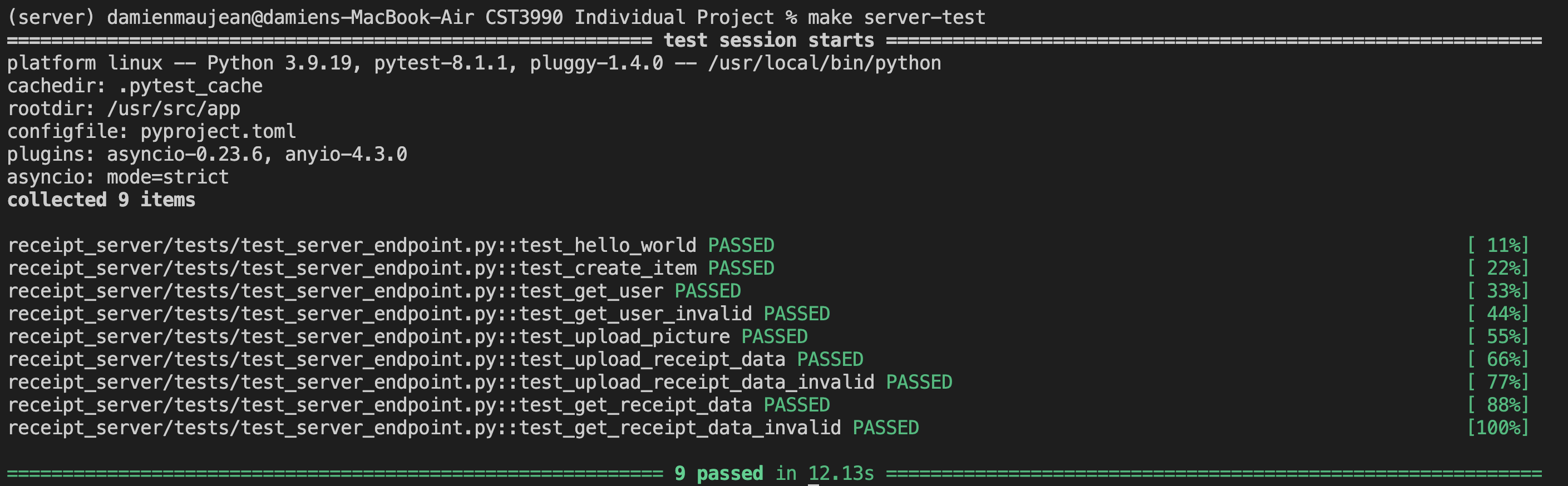
**

Figure 31 - Server test result logs.

### 7.2.2 Receipt Section Detection Model.

For the testing of this Yolo detection model, ensuring the right output of the model is not enough and needs to be able to calculate the approximation accuracy of the prediction. Therefore, since the model prediction area of the receipt, the calculation of the accuracy requires a specific calculation. Inspired by “Information Extraction from Scanned Invoices using Machine Learning, OCR and Spatial Feature Mapping Techniques” research paper which needed to do similar evaluation, it evaluates the percentage of class detected and accuracy of the label when detected. (Darsha, 2023)

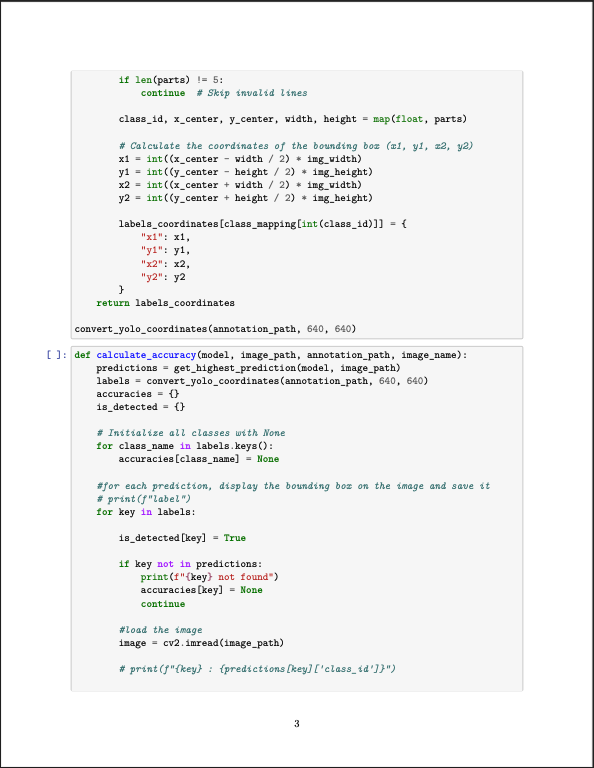
The **percentage of class detected** represent the testing dataset, the percentage where the model did detect an instance of the class on the receipt. It allows to display the model capacity of detecting the presence of each class. The calculation for this value is:

,

This calculation is given for each of the class detection percentage.

As for the **class detection accuracy**, it defines the accuracy of the model prediction area other the area of the label. It allows to give an idea of the precision of the model for when it detects a class. The calculation of this value is:

All the different test and calculation for the accuracy were done using a Jupyter Notebook, which uses these calculations on the training dataset after applying the model on the different test images. An N-Fold test was considered in addition of these calculations, but due to the limitation and difficulties in the gathering of receipts due to their short lifespan and easily become unreadable, it was not completed.



# 8 Results & Analysis

For the evaluation of the model, the dataset uses a larger variety of receipts than the training dataset by using the testing dataset seen in *Figure 11 - Dataset Distribution*. It is composed for the training of most of the type of receipts already seen, such as Super U, London Way, Intermart, Central Water Board, and Mauritius Telecom. In addition, some unseen type of receipts, such as Old Brother Restaurant, Winners, and Emeralda Service Station.

## 8.1 Results

After applying the training on the different codes present in the evaluating Jupyter notebook base on the calculation showed in *Receipt Section Detection Model* section, *Table 5 - Receipt Section Detection Model accuracy result* was generated which summarized the different results from the evaluating Jupyter notebook.

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Number of occurrences not detected | Class detection | Class accuracy |
| Shop Information | 1 | 94% | 96% |
| Total | 2 | 88% | 86% |
| Item Purchase | 0 | 100% | 81% |
| Date | 4 | 77% | 86% |

Table 5 - Receipt Section Detection Model accuracy result`

From this table, we can see that the model identifies in the exception of the date class, correctly detects the different classes with a mean of 89% globally. As for the accuracy of the detection, we can calculate the mean of the accuracy which results with 87%.

Therefore, from these raw values, we can say that, on average, the model detects the different classes of receipts 89% of the time, and when they are detecting the area of the prediction it is 87% accurate.

## 8.2 Analysis

Even if these results show good evaluation of the model, the testing dataset directly impacts the result of these tests due to the small amount and diversity of receipts. Therefore, some deeper analysis could explain the current result displayed for each class in the *Table 5*.

The result for the **Shop Information** class shows good percentage in detection (94%) and accuracy (96%). These results can be explained by the common placement of the shop information among a large variety of receipts, which is usually an isolated set of text centered at the top of the receipt.

As for the **Item Purchase** class, it has the highest detection percentage (100%) but the lowest accuracy percentage (81%). The high detection percentage result was due to the placement of the list of items, which are usually central of the receipt with a repetitive pattern which can be easily detected, as well as it can be voluminous. As for the accuracy of the detection, the size of the item list also alters the accuracy of the prediction because it can represent a large element which can lead to a reduction of the precision. Through the development and multiple tests and retries, the item purchase class extraction of receipts containing a single or small number of lines of product, like in *Figure 32* where the receipt details information were misidentified as the product list, which underlines the difficulty in the accuracy.

Figure 32- Old Brother Restaurant Receipt



The **Date** class presented the model with the most difficulties in the model detection, with 77% with a correct accuracy of 89%. This low percentage in the detection can be explained due to the large variety of formats and placements among the different types of receipts. It represents a small area of the receipt, wherein its placement and format may change for every receipt and make its detection difficult. The high accuracy in the prediction area is due to the presence of receipt types common in the training and testing dataset, which if the class is not detected does not negatively affect the accuracy.

As for the **Total** class, it has a detection percentage of 88% with an accuracy percentage of 86%. Like the Date class, the total of the receipt changes a lot in terms of format, as well as the receipt type changes and, therefore, causes complexities as the same reason as the Date class makes the accuracy percentage low.

These results and the multiple tests during the development of the system has showed a compelling capacity of the model to generalize with some real day-to-day examples of receipts for accurate testing of the Date class. This can be explained with the reduction of quality within the inbuild camera feature from the application, which is why I recommend using a picture of the receipt from the gallery instead. On the other hand, it shows a riveting generalization capacity for the Shop Information, Item Purchase, and Total class. The best way to increase the model capacities would be by having a larger and more balanced dataset, which would offer a sizeable variety of receipts, which may increase the global detection and accuracy. However, increasing the dataset may also impact the Date extraction capacities due to its less developed nature than the Shop Information and Item Purchase class.

# 9 Conclusion

In summary, this project of ‘Smart Receipt Management and Extraction’ aims to facilitate the logging of receipts into a digital format to facilitate the storage, and manual extraction process, as well as reduce paper waste and provide useful analytics about provided receipts and purchases. Since the start, the project’s objective is to provide a user-friendly interface for receipt extraction while also storing the data and providing useful analytics to the user about their receipts and payments. Therefore, to develop this system, a literature review of other research working on similar objectives of this project was written to analyze the technologies they used for the user Interface and receipt extraction technologies. While the literature that was researched and used for my project was adequate, furthering my research into other various technologies would had been more beneficial and would have provided more opportunities to further expand my project.

The project structure was designed before the real implementation in order to organize the development of the system. The system was structured into two different components: the server and the mobile application. The mobile application’s purpose is to be the gateway to the system for the user, and allow the user to access, edit and read the analytics from their receipt, all while communicating with the server. The server’s side manages the communication with the database and host the receipt extraction process and the model used within it.

The development of the server was done locally using docker separating the PostgreSQL database and the FastAPI application which managed the different endpoints which composed the core of the server while communicating with the database. The local deployment provided an easier development environment but limited the range (while deployed locally) of the communication to local environment, but this limitation can be removed with further work. As for the receipt extraction process, it is separated into three main components: the receipt section identification model, the text extractor, and the classifier/formatter. The section identifier model is a Yolov8 model train of gathered receipt dataset, which detects the key sections of receipts. The capacity is directly linked to the gathered dataset, which, with more data, would provide better results. Regarding the text extraction, Tesseract OCR were deployed locally in the server to avoid using a third-party API. The final part was the classification and formatting step which should have been designed to be a custom model. This would have worked, but due to developing restrictions, GPT3.5 turbo llm from OpenAI were used to fulfill this task. Since Tesseract OCR were chosen for the system to be third-party independent, but ended up being dependent by using OpenAI service, the usage of more performant OCR through API such as Google Cloud Vision API would provide better results and increase the extraction quality.

The development of the user side mobile application was done using flutter framework coded in dart, which communicates with the server using http protocol. The mobile application is composed of three pages with their own feature. The list page, which retrieves and displays the user’s receipt history, enables the user to edit the data. The camera page allows the user to upload a picture of a receipt for extraction, either by capturing directly through the application camera or from the user’s gallery. A limitation of the camera feature, which allows the user to take a picture, has a reduced quality which directly impacts the result. Therefore, a custom-made camera plugin could upgrade the inbuilt camera feature. The last page is the graph section, which is from the server’s data that generates two graphs for analytical purposes: a line graph and a pie chart. Even though the application is working effectively, the design could be modified to provide a better user experience and make the application more user-friendly.

Overall, the system task to assist the user in the logging and extraction receipt fullfed its role as assistant but has its limitation in its capacities and some feature of the system could have been better without the different constraint during the development of the project. As discussed in *Results & Analysis* section, the model is efficient to accurately detect the different sections of the receipt with similar receipt with similar format from the one in the training dataset but poorly generalize for unseen format and limit the whole extraction feature. The current test executed cover the key feature to ensure the well execution of the system by using unit testing but are segmentate and doesn’t cover a full end-to-end test which would ensure future development and system stability. Another limitation of the system is the local deployment of the server, which enforce communication application-server to be in local network or tough the usage of VPN and the rebuilding of the whole mobile application when the server rebuilds, and change is private IP. As for the classification/formatting feature of the receipt extraction, the usage of OpenAI large language model do its different task correctly but still have his limitation. It makes the system third party dependent, and its efficiency fluctuate with the possibility of “hallucination”. Finally, the User-Interface of the mobile application fullfed its task to allow the user the use the system different feature. Therefore, it could be more “user-friendly” with better design.

Even if the system has some weakness and limitation, most of them can be resolve with more development. The receipt dataset which is used to train and test the yolo model would need to be extend with more receipts with more variety to increase the generation of the section detections. More tests should be done to increase the system stabilities and ensure the well-being of the system while implementing new feature. The server should be deploy on a server with a public IP to increase the user experience of the system and give a world range access to the system. Another needed work would be to explore more technology for the classification and formatting task, to maybe get some better result. Or if this task continues to be execute by a language model, to modify the prompt sent to it to avoid “hallucination” and enforce well response. As for the text extraction task, since the system is already third party dependent, exploring and testing API base OCR could increase the extraction quality and therefore simplify the formatting and classification task. Finally, some more work could be done on the user-interface to offer a better experience.

From what has been written, researched, analyzed and described above, I believe the research project idea holds great value, with further improve and modification, that would greatly benefit future businesses, operations and everyday living. My receipt extraction project is user-friendly, which is what I aimed to provide, as well as offer straightforward options for companies to easily extract data digitally and professionally, without the use of manual labor. I believe my project to be of great use in the future.

# 

# Reference

AI, R. (no date) *Receiptor AI*, *Receiptor AI*. Available at: https://receiptor.ai/ (Accessed: 30 October 2023).

Alabdullah, T. (2019) ‘Management Accounting and Service Companies’ Performance: Research in Emerging Economies’, *Australasian Accounting, Business and Finance Journal*, 13(4), pp. 100–118. Available at: https://doi.org/10.14453/aabfj.v13i4.8.

Benchekroun, O. (2022) ‘Computer Vision for Document Image Analysis and Text Extraction’.

Darsha, W.B. (2023) ‘Information Extraction From Scanned Invoices using Machine Learning, OCR and Spatial Feature Mapping Techniques’.

*General data protection regulation (GDPR) | EUR-Lex* (no date). Available at: https://eur-lex.europa.eu/EN/legal-content/summary/general-data-protection-regulation-gdpr.html (Accessed: 13 April 2024).

Jiao, B.Z.\*, Evan Frick \*.,. Tianhao Wu \*.,. Hanlin Zhu and Jiantao (2023) *Starling-7B: Increasing LLM Helpfulness & Harmlessness with RLAIF*. Available at: https://starling.cs.berkeley.edu (Accessed: 11 April 2024).

Lenzmann, O. (2024) ‘Mastering LLMs for Complex Classification Tasks’, *Medium*, 15 January. Available at: https://medium.com/@olaf.lenzmann/mastering-llms-for-complex-classification-tasks-64f0bda2edf3 (Accessed: 11 April 2024).

Odd, J. and Theologou, E. (2018) ‘Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms’.

*Open Source Data Labeling* (no date) *Label Studio*. Available at: https://labelstud.io/ (Accessed: 11 April 2024).

*Roboflow* (no date). Available at: https://app.roboflow.com/ (Accessed: 11 April 2024).

*Transform Documents into Actionable Data in Seconds using Veryfi OCR API* (no date) *Veryfi*. Available at: https://www.veryfi.com/ (Accessed: 30 October 2023).

Ultralytics (no date) *Ultralytics*. Available at: https://docs.ultralytics.com/ (Accessed: 11 April 2024).

*What Is the Waterfall Methodology? (Definition + Phases) | Built In* (no date). Available at: https://builtin.com/software-engineering-perspectives/waterfall-methodology (Accessed: 17 April 2024).

Wu, W. (2018) ‘React Native vs Flutter, cross-platform mobile application frameworks’.

# Appendix

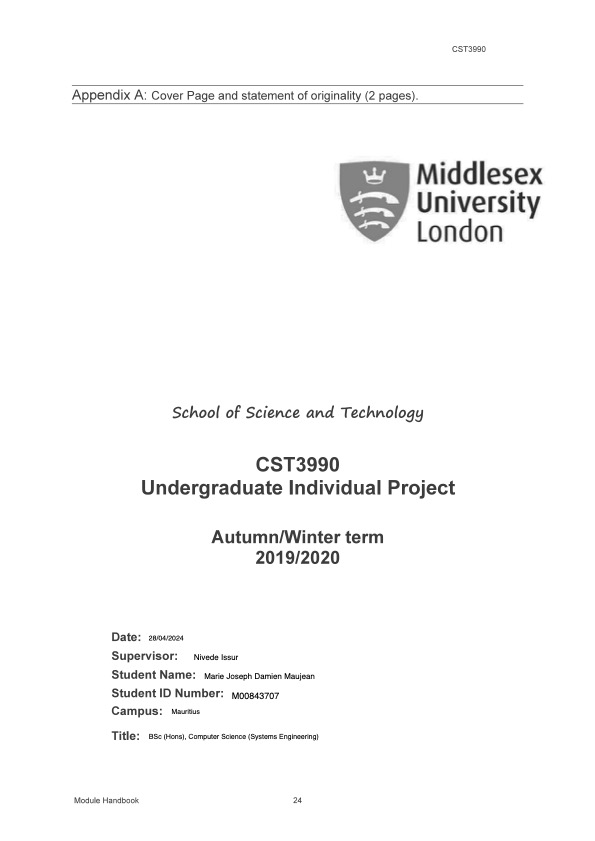
|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:receipt\_extractor/widgets/camera.dart';  import 'package:http/http.dart';  import 'dart:io';  import 'package:image\_picker/image\_picker.dart';  import 'package:receipt\_extractor/widgets/navigation\_bar.dart';  import 'package:receipt\_extractor/classes/data\_service\_class.dart';  import 'package:receipt\_extractor/globals.dart' as globals;  import 'package:receipt\_extractor/widgets/popup.dart';  class CameraPage extends StatefulWidget {  final ImagePicker picker;  final DataService dataService;  final XFile? initialImageFile; // Optional initial image file  CameraPage({Key? key, ImagePicker? picker, DataService? dataService, this.initialImageFile})  : this.picker = picker ?? ImagePicker(),  this.dataService = dataService ?? DataService(),  super(key: key);  @override  \_CameraPageState createState() => \_CameraPageState();  }  class \_CameraPageState extends State<CameraPage> {  XFile? \_imageFile;  double \_imageAspectRatio = 1; // Default aspect ratio  bool \_isUploading = false;  @override  void initState() {  super.initState();  if (widget.initialImageFile != null) {  \_imageFile = widget.initialImageFile;  }  }  Future<void> \_setImageFile(XFile file) async {  final decodedImage = await decodeImageFromList(await file.readAsBytes());  setState(() {  \_imageFile = file;  \_imageAspectRatio = decodedImage.width / decodedImage.height;  });  }  void \_showCameraPopup() {  showModalBottomSheet(  context: context,  isScrollControlled: true,  builder: (BuildContext context) {  return FractionallySizedBox(  key: const Key('Camera Popup'),  heightFactor: 0.8,  child: CameraPopupWidget(  key: const Key('Camera Popup Widget'),  onPictureTaken: (XFile file) {  \_setImageFile(file);  Navigator.pop(context);  },  ),  );  },  );  }  Future<void> \_pickImageFromGallery() async {  try {  final pickedFile = await widget.picker.pickImage(source: ImageSource.gallery);  if (pickedFile != null) {  \_setImageFile(pickedFile);  }  } catch (e) {  print('Failed to pick image: $e');  }  }  Future<void> \_uploadAndSendImage() async {  setState(() {  \_isUploading = true;  });  try {  Response response = await DataService().sendPicture(\_imageFile, globals.user\_id);  if (response.statusCode == 200) {  print('Picture uploaded');  showPopup(context, "Upload Picture", response.body, false);  } else {  throw Exception('Failed to upload picture');  }  } catch (e) {  } finally {  setState(() {  \_isUploading = false;  });  }  }  @override  Widget build(BuildContext context) {  return Scaffold(  key: const Key('Camera Page'),  appBar: AppBar(title: const Text('Take a Picture')),  body: Center(  child: Column(  mainAxisAlignment: MainAxisAlignment.center,  children: <Widget>[  Expanded(  child: \_imageFile == null  ? const Center(  key: Key('No Image Selected Message'),  child: Text(  "No image selected.",  style: TextStyle(  fontSize: 16,  fontWeight: FontWeight.bold,  ),  ),  )  : AspectRatio(  aspectRatio: \_imageAspectRatio,  child: Image.file(File(\_imageFile!.path), fit: BoxFit.contain),  ),  ),  Row(  mainAxisAlignment: MainAxisAlignment.center,  children: [  ElevatedButton(  key: const Key('Open Camera Button'),  onPressed: \_showCameraPopup,  style: ElevatedButton.styleFrom(  minimumSize: const Size(120, 60),  shape: const RoundedRectangleBorder(  borderRadius: BorderRadius.zero,  ),  padding: const EdgeInsets.all(16),  ),  child: const Text('Open Camera'),  ),  ElevatedButton(  key: const Key('Open Gallery Button'),  onPressed: \_pickImageFromGallery,  style: ElevatedButton.styleFrom(  minimumSize: const Size(120, 60),  shape: const RoundedRectangleBorder(  borderRadius: BorderRadius.zero,  ),  padding: const EdgeInsets.all(16),  ),  child: const Text('Open Gallery'),  ),  if (\_imageFile != null)  ElevatedButton.icon(  key: const Key('Extract Button'),  onPressed: \_isUploading ? null : \_uploadAndSendImage,  icon: \_isUploading  ? const SizedBox(  height: 20.0,  width: 20.0,  child: CircularProgressIndicator(  strokeWidth: 2.0,  ),): const Icon(Icons.upload),  label: const Text('Extract'),  style: ElevatedButton.styleFrom(  minimumSize: const Size(120, 60),  shape: const RoundedRectangleBorder(  borderRadius: BorderRadius.zero,  ),  padding: const EdgeInsets.all(16),  ),  ),  ],  ),  ],  ),  ),  bottomNavigationBar: CustomBottomNavigationBar(selectedIndex: 1),  );  }  } |

Appendix 1 - Camera Page code

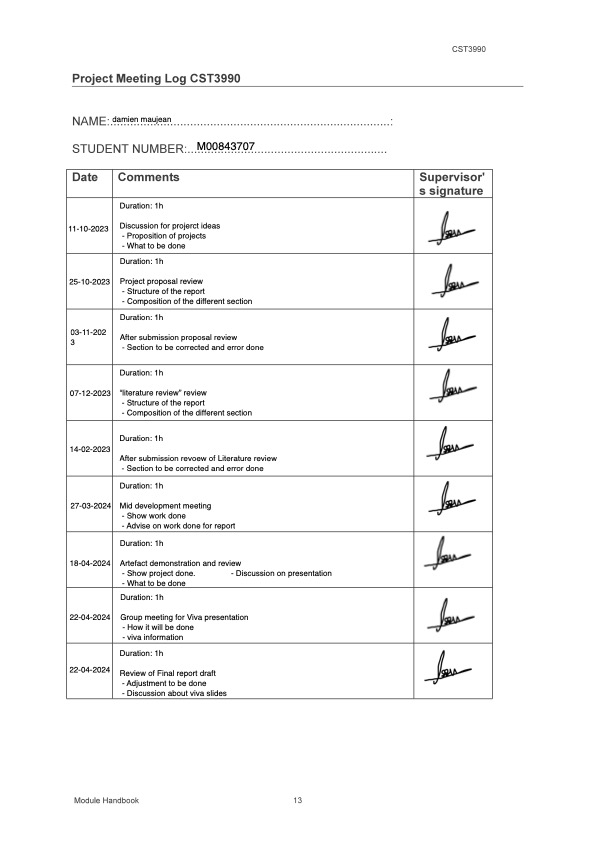
|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:fl\_chart/fl\_chart.dart';  import 'package:receipt\_extractor/widgets/navigation\_bar.dart';  import 'package:fl\_chart/fl\_chart.dart' as fl\_Chart;  import 'package:intl/intl.dart';  import 'package:receipt\_extractor/widgets/analitics/pie\_chart.dart';  import 'package:receipt\_extractor/classes/receipt\_class.dart';  import 'package:receipt\_extractor/classes/data\_service\_class.dart';  import 'package:receipt\_extractor/widgets/analitics/line\_chart.dart';  import 'package:receipt\_extractor/globals.dart' as globals;  class GraphPage extends StatefulWidget {  final DataService dataService;  GraphPage({Key? key, DataService? dataService})  : dataService = dataService ?? DataService(),  super(key: key);  @override  \_GraphPageState createState() => \_GraphPageState();  }  class \_GraphPageState extends State<GraphPage> with SingleTickerProviderStateMixin {  late TabController \_tabController;  late Future<List<Receipt>> futureReceipts;  late List<FlSpot> spots;    @override  void initState() {  super.initState();  \_tabController = TabController(length: 2, vsync: this);  futureReceipts = widget.dataService.fetchReceipts(globals.user\_id);  }  @override  void dispose() {  \_tabController.dispose();  super.dispose();  }  Widget buildAnimatedLineChart(List<Receipt> data, List<FlSpot> spots) {  Map<DateTime, double> groupByMonth(List<Receipt> receipts) {  Map<DateTime, double> monthlyTotals = {};  for (var receipt in receipts) {  DateTime month = DateTime(receipt.date.year, receipt.date.month);  if (!monthlyTotals.containsKey(month)) {  monthlyTotals[month] = 0;  }  monthlyTotals[month] = monthlyTotals[month]! + receipt.total;  }  return monthlyTotals;  }  List<Receipt> filterLatestThreeMonths(List<Receipt> receipts) {  receipts.sort((a, b) => b.date.compareTo(a.date));  var uniqueDates = receipts.map((e) => DateTime(e.date.year, e.date.month)).toSet().toList();  uniqueDates.sort((a, b) => b.compareTo(a));  uniqueDates = uniqueDates.take(4).toList();  return receipts.where((receipt) {  var receiptMonth = DateTime(receipt.date.year, receipt.date.month);  return uniqueDates.contains(receiptMonth);  }).toList();  }  List<Receipt> latestData = filterLatestThreeMonths(data);  Map<DateTime, double> monthlyTotals = groupByMonth(latestData);  List<FlSpot> spots = monthlyTotals.entries.map((entry) {  return FlSpot(entry.key.millisecondsSinceEpoch.toDouble(), entry.value);  }).toList();  this.spots = spots;  return Container(  padding: const EdgeInsets.all(50),  width: double.infinity,  height: double.infinity,  child: fl\_Chart.LineChart(  LineChartData(  lineBarsData: [  LineChartBarData(  spots: spots,  isCurved: false,  dotData: FlDotData(  show: false,  ),  color: Colors.red  ),  ],  borderData: FlBorderData(  border: const Border(bottom: BorderSide(), left: BorderSide())),  gridData: FlGridData(show: false),  titlesData: FlTitlesData(  bottomTitles: AxisTitles(sideTitles: \_bottomTitles),  leftTitles: AxisTitles(sideTitles: \_leftTitles),  topTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),  rightTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),  ),  ),  ),  );  }  SideTitles get \_bottomTitles => SideTitles(  showTitles: true,  reservedSize: 30,  getTitlesWidget: (value, meta) {  bool isCloseToSpot(double spotValue, double checkValue) {  return (spotValue - 300000000.0 <= checkValue) && (checkValue <= spotValue + 300000000.0);  }  bool hasCloseSpot = spots.any((spot) => isCloseToSpot(spot.x, value));  if (hasCloseSpot) {  final DateTime date = DateTime.fromMillisecondsSinceEpoch(value.toInt());  final String formattedDate = DateFormat('yyyy-MM').format(date);  return Padding(  padding: const EdgeInsets.symmetric(horizontal: 4),  child: Text(  formattedDate,  style: const TextStyle(  fontWeight: FontWeight.bold,  fontSize: 8,  ),  ),  );  } else {  return const Text('');  }  },  );  SideTitles get \_leftTitles => SideTitles(  showTitles: true,  reservedSize: 30,  getTitlesWidget: (value, meta) {  return Text(  value.toStringAsFixed(2),  style: const TextStyle(  fontWeight: FontWeight.bold,  fontSize: 12,  ),  textAlign: TextAlign.right,  );  },  );  @override  Widget build(BuildContext context) {  return Scaffold(  key: const Key('Graph Page'),  appBar: AppBar(  flexibleSpace: Column(  mainAxisAlignment: MainAxisAlignment.end,  children: [  TabBar(  key: const Key('Tab Bar'),  controller: \_tabController,  tabs: const [  Tab(key: Key("Tab 1"), text: 'Line Chart'),  Tab(key: Key("Tab 2"), text: 'Pie Chart'),  ],  )  ],  ),  ),  body: FutureBuilder<List<Receipt>>(  future: futureReceipts,  builder: (context, snapshot) {  spots = [];  if (snapshot.connectionState == ConnectionState.waiting) {  return const Center(child: CircularProgressIndicator());  } else if (snapshot.hasError) {  return Column(  key: const Key('Error Message'),  mainAxisAlignment: MainAxisAlignment.center,  children: [  const Icon(  Icons.error\_outline,  size: 48,  color: Colors.red,  ),  const SizedBox(height: 16),  Padding(  padding: const EdgeInsets.symmetric(horizontal: 16),  child: Text(  'Error: ${snapshot.error}',  style: const TextStyle(  fontSize: 16,  fontWeight: FontWeight.bold,  color: Colors.red,  ),  ),  ),  ],  );  } else if (snapshot.hasData && snapshot.data!.isNotEmpty) {  return TabBarView(  key: const Key('Tab Bar View'),  controller: \_tabController,  children: [  AnimatedLineChartWidget(data: snapshot.data!),  BarChartWithSelector(data: snapshot.data!),  ],  );  } else {  return const Center(  key: Key('No Data Message'),  child: Text(  'No data available',  style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold)  ),  );  }  },  ),  bottomNavigationBar: CustomBottomNavigationBar(selectedIndex: 2),  );  }  } |

Appendix 2 - Graph Page Code

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:receipt\_extractor/widgets/navigation\_bar.dart';  import 'package:receipt\_extractor/classes/receipt\_class.dart';  import 'package:receipt\_extractor/classes/data\_service\_class.dart';  import 'package:receipt\_extractor/globals.dart' as globals;  import 'package:receipt\_extractor/widgets/popup.dart';  class ReceiptsPage extends StatefulWidget {  final String? userId; // Optional user ID  final DataService dataService; // Data service dependency  // Non-const constructor  ReceiptsPage({Key? key, this.userId, DataService? dataService})  : dataService = dataService ?? DataService(),  super(key: key);  @override  \_ReceiptsPageState createState() => \_ReceiptsPageState();  }  class \_ReceiptsPageState extends State<ReceiptsPage> {  late Future<List<Receipt>> futureReceipts;  @override  void initState() {  super.initState();  String userId = widget.userId ?? globals.user\_id;  futureReceipts = widget.dataService.fetchReceipts(userId);  }  void refreshReceipts() {  setState(() {  // Refresh with the current userId in use  String userId = widget.userId ?? globals.user\_id;  futureReceipts = widget.dataService.fetchReceipts(userId);  });  }  @override  Widget build(BuildContext context) {  return Scaffold(  key: const Key('Receipts Page'),  appBar: AppBar(  actions: [  IconButton(  icon: const Icon(Icons.refresh),  onPressed: refreshReceipts,  ),  IconButton(  key: const Key('Add Button'),  icon: const Icon(Icons.add),  onPressed: () {  print("add receipt");  showPopup(context, "Add Receipt", "", true);  refreshReceipts();  },  ),  ],  title: const Text('Receipts'),  ),  body: FutureBuilder<List<Receipt>>(  future: futureReceipts,  builder: (context, snapshot) {  if (snapshot.connectionState == ConnectionState.waiting) {  return const Center(child: CircularProgressIndicator());  } else if (snapshot.hasError) {  return Column(  key: const Key('Error Message'),  mainAxisAlignment: MainAxisAlignment.center,  children: [  const Icon(Icons.error\_outline, size: 48, color: Colors.red),  const SizedBox(height: 16),  Padding(  padding: const EdgeInsets.symmetric(horizontal: 16),  child: Text('Error: ${snapshot.error}', style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold, color: Colors.red)),  ),  ],  );  } else if (snapshot.hasData && snapshot.data!.isNotEmpty) {  return ListView.builder(  itemCount: snapshot.data!.length,  itemBuilder: (context, index) {  var receipt = snapshot.data![index];  return ListTile(  key: Key("row $index"),  title: Text(receipt.shop\_name),  subtitle: Text('Total: ${receipt.total.toString()}'),  trailing: Row(  mainAxisSize: MainAxisSize.min,  children: [  Text(receipt.date.toLocal().toString().split(' ')[0]),  const SizedBox(width: 8),  const Icon(Icons.edit\_note),  ],  ),  onTap: () {  print("edit receipt: ${receipt.toString()}");  showPopup(context, "Edit Receipt", receipt.toString(), false);  refreshReceipts();  },  );  },  );  } else {  return const Center(  key: Key('No Receipts Message'),  child: Text("No receipts found. Add a new receipt.", style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold)),  );  }  },  ),  bottomNavigationBar: CustomBottomNavigationBar(selectedIndex: 0),  );  }  } |

Appendix 3 - List Page Code

Appendix 4 - Meeting Log Part 1



Appendix 5 - Meeting Log Part 2