Soma Benfell

sbenfell8gff@student.attleboroughacademy.org

Abstract

Documentation and source for the OneStop Stock Monitoring Assistant (OSSMA) developed for my Non-Examination Assessment  
  
GitHub source control:   
https://github.com/sbenf999/Stock-Monitoring-Assistant



Stock monitoring ASSIsTANT

AQA A-Level Computer Science NEA

Table of Contents

[Analysis 3](#_Toc194654793)

[Outlining the problem 3](#_Toc194654794)

[Lifecycle of a product at OneStop 4](#_Toc194654795)

[Has anyone looked at this before? 5](#_Toc194654796)

[System one – *Inventoria* Inventory Software (NCH Software) 5](#_Toc194654797)

[System two – *InvenTree*- Open-Source Inventory Management System 5](#_Toc194654798)

[System three – *SubVentory* – Stock control system for subway 5](#_Toc194654799)

[Research 6](#_Toc194654800)

[Primary Research 6](#_Toc194654801)

[Primary Research Methods - Interview 6](#_Toc194654802)

[Results and analysis 7](#_Toc194654803)

[Extra considerations 8](#_Toc194654804)

[User privileges based on interview 8](#_Toc194654805)

[MoSCoW Analysis 9](#_Toc194654806)

[Must – functional things the program must do 9](#_Toc194654807)

[Should – usability considerations for *how* the program should do the things 10](#_Toc194654808)

[Could – desirable functional or usability features the program *could* do in the future 10](#_Toc194654809)

[Won’t – things that are out of the scope of the project 11](#_Toc194654810)

[Secondary Research 12](#_Toc194654811)

[Domain specific research 12](#_Toc194654812)

[Programming Language Research 14](#_Toc194654813)

[Python library/package usage table of references 15](#_Toc194654814)

[Similar Systems Analysis 16](#_Toc194654815)

[Design section 18](#_Toc194654816)

[ER Diagram 18](#_Toc194654817)

[Use case diagrams and use case descriptions 20](#_Toc194654818)

[Data dictionary 27](#_Toc194654819)

[Users table 27](#_Toc194654820)

[Product table 27](#_Toc194654821)

[Suppliers table 28](#_Toc194654822)

[Waste table 28](#_Toc194654823)

[Stock level table 28](#_Toc194654824)

[Stock level history table 29](#_Toc194654825)

[Weekly reports record table 29](#_Toc194654826)

[UML class diagram 31](#_Toc194654827)

[DB Handling classes 31](#_Toc194654828)

[Process classes 32](#_Toc194654829)

[Main.py classes 32](#_Toc194654830)

[Initial UI wireframe diagram (Kotlin/Java – prior lang. change) 33](#_Toc194654831)

[Final UI wireframe diagram (Python – after lang. change) 34](#_Toc194654832)

[Discussion and changes 34](#_Toc194654833)

[Updated wireframe designs 34](#_Toc194654834)

[Technical implementation 36](#_Toc194654835)

[Resources 36](#_Toc194654836)

[Code 37](#_Toc194654837)

[Project structure contents page 37](#_Toc194654838)

[Source 39](#_Toc194654839)

[Requirements met 110](#_Toc194654840)

[Testing 112](#_Toc194654841)

[Functionality testing 112](#_Toc194654842)

[Usability testing 115](#_Toc194654843)

[Evaluation 118](#_Toc194654844)

[Final thoughts 124](#_Toc194654845)

[Success of my project 124](#_Toc194654846)

[References 125](#_Toc194654847)

# Analysis

## Outlining the problem

Many 6th form students have a part-time job during their studies to earn some extra spending money whilst on their 2-year courses. As one such student, I applied to work the tills at the local *OneStop* store in Attleborough, a convenience chain in the UK that sells groceries, newspapers and more. Their mission, as stated on their website, is to “provide convenience and accessibility for all of [our] customers” - (OneStop Ltd, 2019).

Having worked on the shop floor consistently for almost a year (with breaks in between for exams and such), I have become accustomed to the process by which deliveries are made, stock is counted, and waste dealt with. The manager runs an efficient day to day business, with time maximized to be used in the best way possible. However, there is one issue that he (the boss) has run into, that he has not been able to fix. The shop purchases stock from one of two groups of suppliers, as can be seen in **Figure 1**.

A diagram of a business

Description automatically generatedFirstly, most products are supplied by *OneStop* themselves, and often include items you might find in your local *Tesco’s*, simply rebranded, as they fall under the same parent company. This stock is managed using a proprietary stock management system already supplied to franchisees by *OneStop.* This makes the process smooth and reasonably more simple for a staff member who may be tasked with such things such as recording stock.

Figure 1

On the other hand, the second group of stock consists of suppliers who the manager has chosen themselves. A good example could be local sourced goods or a baked good company that delivers bread and pastries. These suppliers do not have a stock management system for the Attleborough based *OneStop* to operate, and so all stock counting is done by hand on paper, alongside any notes of waste and previous deliveries. This creates certain issues in the day to day running of the store:

* Items supplied by local businesses are not logged into the database(s) used by the proprietary *OneStop* stock control system and either cannot be added to it or expertise required to do so is not possessed by the manager of the Attleborough store and/or affiliated members of staff.
* Due to the above reason, stock counting is performed by hand, using a pen and paper. This is:
  + Inefficient
  + Difficult to track changes and keep a consistent record of data
  + Time could be spent on more important parts of the business

As there is no system in place already to solve this issue, I have been tasked with finding a solution to decrease the amount of time that staff must spend counting this stock, and for the manager to know when to re-order the items based on how many are on the shop floor, including any excess stock. By removing this process entirely, a large amount of time and effort can be saved, which can be allocated to more important jobs and tasks around the shop. To demonstrate this problem visually, I have created a second rich picture diagram to show what the current life cycle of a product is when it enters the shop, in **Figure 2.** From a generalized perspective, the best approach to this problem would be taking out the pen and paper stage of the process, potentially by creating an application to streamline the process. To work in a A diagram of a product

Description automatically generated*OneStop* store, where alcohol and products are sold, a staff member must be over the age of 18. Even though the technical expertise of a staff member to operate such an application is not very high, they may have varied technical knowledge. Therefore, the design of the solution must be user-friendly, and menu driven to provide a consistent user experience throughout its usage.

### Lifecycle of a product at OneStop

At *OneStop*, the typical product enters a product lifecycle, typically being ordered when it meets a re-order threshold (the amount of stock for said product has dropped below a certain point and needs to be reordered to prevent missing products on a shelf/this occurring). Once it is ordered, there is a lead time, the time it takes from ordering to it being shipped and being put out on shelves on the shop floor. From here, a product can go one of two ways: it may be purchased by a customer, for which amount the stock level will be decreased by the amount purchased. On the other hand, a product may become damaged. This often happens during deliveries or stocking shelves, where an employee may drop the product or somehow or otherwise damage it. Additionally, customers may accidentally damage a product, such as by dropping them. This often happens with items such as beverages. When this happens, they need to be recorded as waste, as they can no longer be sold due to their condition. This “status” as being waste is then recorded on paper by hand and deducted from the net profit as an expense to OneStop, as the product cannot be sold. This lifecycle is shown in the below flowchart. As you can

|  |  |
| --- | --- |
| A diagram of a company  AI-generated content may be incorrect. | see in the flow chart on the left, if a product is damaged, it gets placed in the managers office and noted down as a waste item. This is then accounted for, the next time the manager is in the office. It is disposed of appropriately, the details of why it is waste being noted, the stock level being decreased, and therefore the profits also. |

## Has anyone looked at this before?

### *A screenshot of a software Description automatically generated*System one – *Inventoria* Inventory Software (NCH Software)

*Inventoria* is designed to be as intuitive to use as possible, so after a quick instillation, “*you’ll be streamlining your inventory process within minutes*” (NCH Software, 2008).

The inventory software is stated to be:

* Easily organize and keep track of inventory.
* Never run out of stock with warnings and reports
* Installs and is running in just minutes.

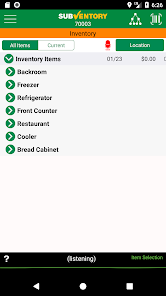
*Inventoria* inventory software is a good example to analyse the simplicity of a stock control system. Whilst it may not look modern and flashy, it performs all basic functions in a satisfactory manner.

Figure 2 (NCH Software, 2024)

### System two – *InvenTree*- Open-Source Inventory Management System

“*InvenTree is an open-source inventory management system which provides intuitive parts management and stock control. A wide range of features makes InvenTree the perfect choice for businesses and hobbyists alike. InvenTree is designed to be extensible, and provides multiple options for integration with external applications or addition of custom plugins*” - (InvenTree, 2024)

Figure 5 (InvenTree, 2024)



### System three – *SubVentory* – Stock control system for subway

“*SubVentory is the fastest and most accurate way for any restaurant operator to capture their week-ending inventory, deliveries, file QualityNet Pro complaints, place new distributor orders, and manage withdrawals and recalls. The app helps eliminate errors before they happen and saves hours in labour each week*” - (ZippyZum, 2024)

*SubVentory* was used in OneStop when they still had a Subway to manage (that has since been closed to unrelated reasons).

Figure 6 (ZippyZum, 2024)

# Research

## Primary Research

### Primary Research Methods - Interview

|  |  |  |
| --- | --- | --- |
| *What I need to know* | *Source of information* | *Research method and justification* |
| *How is stock counted at the moment?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Observation*  Justification: This will allow me to gain understanding around the current process involved in stock counting, which I can adapt to the project. |
| *Which staff carry out the stock counting?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Questionnaire*  Justification: By knowing what type of staff carry out the stock counting, I can create access levels based on their roles within the shop to better reflect their job roles within the application. |
| *How frequently is this done?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: The frequency at which stock counts need to take place is vital in knowing how often to notify the user to carry out a stock take within the app, to keep an up-to-date database. |
| *What platform/device must the application be designed for?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: Knowing the platform that the application is intended for in important when choosing a programming language and/or framework, as compatibility with certain operating systems is sometimes a pitfall of one tool compared to another. |
| *How easy to use does the application need to be?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Questionnaire*  Justification: Staff may vary in their technical know-how, especially with how age may differ. Whilst younger staff could find it easier to use more complex application, older staff may find it more challenging. For this reason, ease of use is an important factor to know when designing parts of the project such as the user interface. |
| *Are there any preferences on the looks on the application (e.g., house style)?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Questionnaire*  Justification: The client may prefer a specific look and feel to their application, alongside customisation this could make the integration to a new system easier for staff members who have already learned to use a differing system, such as the one provided by *OneStop* already*.* |
| *Does the system need a secure login system for staff?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: Needing a login system depends on how secure the client wants the data to be. Without a login system, data is susceptible to being used by people outside of the business. This also integrates with the access level of each staff member, to accurately portray their real-life responsibilities within the application and prevent people from accessing sensitive data when they shouldn’t. |
| *Should the application send automated reminders on when stock needs to be ordered depending on data found in the database concerning trends in purchases of said stock?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: Automated reminders could be helpful to the client, as it allows them to forecast when a delivery may need to be scheduled, or any other events requiring their attention that is happening within the application. Further examples could be a certain number of incorrect password attempts or that a new delivery has been recorded. |
| *Should database data be local to the device or be stored on a server that the device connects to?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: This detail is important to know. Whilst it is true that the database *could* be hosted on the same device as on which the application is running, it introduces a single point of failure: if the device were to break or malfunction, the data hosted on the database would be unable to be accessed by anyone, as the device was also acting as the server. For this reason, it would be advisable to have a separate computer running the server. Firstly, this ensure that the data is always accessible even if the device on which the application is installed is powered down – (also this would require 24/7 uptime of the server computer). It also means that it is easier to implement the safe storage of any data with periodic backups. |
| *Should reports be able to be printed off, showing a breakdown of the past n number of days’ worth of sales?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: Whilst this data can be shown in app and potentially through a notification system, hard copies of important information can be useful for filing away data, additionally allowing for analysis away from the application. |
| *Should the application be able to print out an export of the analytics?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: As the clients have printers in their office, they may wish to visualise the graphs or any data also on data, allowing them to, for example, annotate or discuss ideas. By allowing them to print/export the graphs, this facilitates this need. |
| *Is there a computer that any needed database software can be run on, something that is powered on 24/7?* | OneStop Staff (Manager, assistant manager and till workers) | Research method: *Interview*  Justification: This computer would be needed if the client chooses to have a remote database alongside the application, to have a 24/7 uptime. |

### Results and analysis

|  |  |
| --- | --- |
| *Question* | *Answer / gathered data* |
| *How is stock counted at the moment?* | Stock is counted upon delivery and at subsequent weekly intervals. However, this may vary due to how fast a product may sell. |
| *Which staff carry out the stock counting?* | Most if not all staff will carry out stock counting. This is generally done on delivery days, as not all products arrive at the same time. An example of this, is that *vape* deliveries may arrive on Thursday, and get counted then, however *sweets* may be delivered on Tuesdays, and therefore get counted on a Tuesday. |
| *How frequently is this done?* | The number of times stock counts are carried out varies based on products but is generally done upon a delivery, which will occur sometimes once a week for products which sell fast. |
| *What platform/device must the application be designed for?* | The client would prefer the application to be able to be used on a mobile device, such as a smartphone or tablet. This ensures that stuff completing tasks such as a stock count can navigate around the store with ease. |
| *How easy to use does the application need to be?* | The client would prefer the application to be as easy to use as possible. This means an appearance which is easy both on the eyes, and to navigate, ensuring that all staff can use the app efficiently and require minimal training to do so (*emphasis on a “5-minute chat and go” as described by the client*) |
| *Are there any preferences on the looks on the application (e.g., house style)?* | Preferences in “house style” are bare-bones as possible to make the app intuitive to use. However, it is in consideration to add a “theme” setting so the user can select a style of their choice, the default being the “barebones” look. A theme could simply change the colour of buttons or the background, with choices being red, green and blue. |
| *Does the system need a secure login system for staff?* | A logon system would be useful to prevent unauthorised users from accessing the data on the application. This could be made more technical my including user access levels. For example, the manager and assistant manager could edit details stored about the suppliers, while this feature may be locked for till operators. Furthermore, having user profiles allows certain tasks to be attributed to staff who carry out the task. |
| *Should the application send automated reminders on when stock needs to be ordered depending on data found in the database concerning trends in purchases of said stock?* | The users / the work group chat should be notified by the application when re-orders need to take place. This will help ensure that fast selling products don’t sell out and leave gaps on shelves. Additionally, a *minimum stock level* should be considered, whereby a warning should be sent notifying if the stock count for a specific item is close to falling below this margin. This could be based on a calculated sale rate and then extrapolated to predict when it may happen. |
| *Should database data be local to the device or be stored on a server that the device connects to?* | Data should be stored on a server that is located in the OneStop office. This means that if the device that the application is installed on were to break, then important data would not be lost. Additionally, this means that the manager can access the data at different times and not have to using the device in order to analyse it or edit any settings. |
| *Should reports be able to be printed off, showing a breakdown of the past n number of days’ worth of sales?* | This feature would be useful as it allows OneStop to keep a physical record, *n* days’ worth of sales. Additionally, it can present a simple overview of sales. |
| *Should the application be able to print out an export of the analytics?* | Whilst it is not so important to print out a copy of the analytics/analysis of sales data, it would be useful to see and highlight the fast and slow selling products. However, a digital analysis section could be implemented in the app. |
| *Is there a computer that any needed database software can be run on, something that is powered on 24/7?* | Yes, there is a tower personal computer that is turned on 24/7 that can be used to run the database software needed, namely MySQL connector and MySQL Workbench. These need to be installed and running before the app is tested to ensure smooth functionality. |

### Extra considerations

|  |  |
| --- | --- |
| *Barcode creator* | The app should be able to generate barcodes upon the creation of a new product in app, which can then be printed off/sent to a label maker and stuck on the relevant product(s). |
| *Barcode scanner* | It may be useful to be able to scan barcodes within the app, such as being able to access useful data about a product in a manner that is fast. However, this may require a specialised device that has a built-in barcode scanner. Devices such as these do exist but are not commonplace. |

### User privileges based on interview

|  |  |
| --- | --- |
| *User access level* | *Meaning* |
| *1* | *Administrator privileges – access to any and all functionality of the program (default privileges*\**)* |
| *2* | *Default privileges minus ["Data view", "Weekly report", "Settings"]* |
| *3* | *Default privileges minus ["Add product", "Add supplier", "Data view", "Weekly report", "Settings", "Waste"]* |

\*Default privileges include: *["Home", "Record a delivery", "Stock counting", "Data view", "Add product", "Add supplier", "Waste", "Weekly report", "Settings"]*

## MoSCoW Analysis

### Must – functional things the program must do

* **M1**: The application must be able to run a counting process where a staff member goes around the shop and tallies each product that is needed and then records this information (*stock\_count)* in the product database, updating previous stock counts.
  + **M1A**: The record stock delivery functionality must have an easy to use and understand UI and enough labels to accurately describe what each input is need for.
  + **M1B**: Staff should be able to record a stock count and then the program update the respective stock levels for each product
  + **M1C**: Stock levels must be able to be displayed in a legible manner
  + **M1D**: The change in stock level for the current week should be displayed/shown when a specific product entry is clicked on
  + **M1E**: Any changes to the stock level of a product need to also be stored in a stock level history table, so that the data can be analysed for the weekly report.
* **M2:** The application must be able to create products
  + **M2A**:Based on the user access level (shown in [User privileges based on interview](#_User_privileges_based)), staff should be able to create and edit details about *products* (*product\_id, supplier\_id, product\_name, product\_description, product\_pack\_size, product\_weight, product\_barcode, product\_price*) within the “product” database, allowing for accurate information at all times.
  + **M2B**: The new product should be added to the database table for products, and any necessary UI elements that rely on the products to be displayed need to be updated in order to show the entry of the new product into the system.
* **M3:** The application must be able to create suppliers
  + **M3A**:Based on the user access level, staff should be able to create and edit details about *suppliers* within the “suppliers” database, allowing for accurate information at all times (stock\_id, product\_id, stock\_count, minimum\_stock\_level, reorder\_level, lastDelivery)
  + **M3B**: The functionality should update any areas where suppliers need to be selected, such as the data view functionality or where there are any lists that use supplier names, so that data is consistent throughout the system without needing to close and re-open the application.
* **M4:** The application must be able to record stock deliveries
  + **M4A**: The record stock delivery functionality must have an easy to use and understand UI, such as having a date picker to save the user have to input a date manually and enough labels to accurately describe what each input is need for.
  + **M4B**: On days when there are stock deliveries, the app must be able to record what has been delivered and in what quantity. This should automatically update stock levels for the correct products.
* **M5:** The application must present analytics/state information on sales
  + **M5A**: The application must display stock counts and any necessary information about products, suppliers and waste in a legible manner. A useful way to present this would be in a table, with a search function that highlights the search term if it exists, and a button appears which allows you to see the graph detailing the trend in stock level for that product (if the user is on the products tab for e.g.)
* **M6:** The application must be able to record any waste (this includes but is not limited to stolen, perished and damaged items)
  + **M6A**: There must be a separate tab in the application for recording any waste. This should include a “*resolvement*” status (for example, green being resolved, red being unresolved and yellow being in progress)
  + **M6B**: Waste should be able to be “*resolved*” (the status of the waste record to be updated to true or false). The user should be able to choose a waste product and update its respective status.
* **M7:** The application must notify you when re-orders are needed
  + **M7A**: Re-orders could be based on a set date or only trigger a notification when a minimum stock level is reached.
  + **M7B**: A notification of this need for a re-order should be sent by email to the default email address.
* **M8:** The application must be able to highlight fast and slow selling items/trends in stock level
  + **M8A**: This information could be highlighted in a weekly report or notification. Allows management to make informed decisions on what product to purchase for the store.
* **M9:** The application must be able to generate a weekly report
  + **M9A**: The program should predict a stock level for each day of the coming week. This prediction should be based on the data of the previous weeks analysis.
  + **M9B**: The program should calculate the revenue, cost of goods sold and then net profit
  + **M9C**: The program should calculate the trend in the weeks stock level for each product using linear regression analysis
  + **M9D**: The user should have the option to automatically email this data to the default email address upon generation of a weekly report
  + **M9E**: The user should have the option to generate a .txt output of the weekly report once it has been generated
  + **M9F**: The user should be able to view previous weekly reports in the database
* **M10**: The application must have a login system, where the user enters their username/unique staff ID and password, and user accounts should be able to be created
  + **M10A**: The login system must have the functionality of being able to change their password. This can be done by entering their old password, the new one, and confirming the new one. If the user has forgotten their password, they can reset it using a recovery key that is generated for them upon account generation.
  + **M10B:** The application should have a user access level-based functionality, so that each user access level corresponds to a certain amount of allowed functionality (see [User privileges based on interview](#_User_privileges_based)).
  + **M10C:** The user must be able to reset their password if they forget it.
  + **M10D**: Admin users must be able to create new user accounts once they have navigated to the settings pane.
* **M11:** The application should provide a visual breakdown of the database tables so that a user can understand what is happening
  + **M11A:** This could include a pie chart with a key detailing all the different database tables present
* **M12**: The application should allow admin users access to a settings pane
  + **M12A**: the admin should be able to create new users
  + **M12B**: The admin should be able to change any environment variables as detailed in [environment Variables](#environmentVariables)
* **M13:** The program must have the ability to send emails (provided it has access to the internet)
  + **M13A:** The program must be able to send emails when necessary to inform staff/admins of any alerts or actions that have occurred in the application.

### Should – usability considerations for *how* the program should do the things

* **S1**: The application should be easy on the eye
  + **S1A**: There shouldn’t be too much information on a page at one time. Information should be presented clearly and effectively. Buttons should be large and colourful, and obvious to what action they perform.
* **S2**: The application should be intuitive
  + **S2A**: A range of staff will be using the application, some more technologically adept than others. Therefore, the program should be as easy to use as possible.
* **S3:** The application should store data on a server local to the business
  + **S3A**: A suitable location could be the office that the manager and assistant manager use
* **S4:** The application should notify staff members through either WhatsApp, email or text messages
  + **S4A**: Notifications should be accessible by the entire staff team. The easiest way to do this would be to automate notifications by sending them to the WhatsApp work group chat which all members of staff are a part of and can access easily, or by sending emails to a default email address or to the user email address once they have logged in.
* **S5:** Consider using a tabulated view-based menu system for easy navigation, allowing for all staff to be able to understand how to use the app without much prerequisite technical knowledge.
  + **S5A**: This ease of usability should also be enhanced by having a vertical navigation bar for the tabulated view with larger buttons, aswell as the built in one provided by the tabulated view in *customtkinter*.
* **S6:** The program should allow a user to update any environment variables
  + **S6A:** The user, if logged in as an admin, should be able to edit/change any environment variables found in [envVarTemplate](#envVarTemplate)
* **S7:** Have autocomplete entries for any search bars to make the program easier to use for staff members
  + **S7A:** This could be useful for auto filling product names etc in the data view UI tab

### Could – desirable functional or usability features the program *could* do in the future

* **C1:** The application could create barcodes.
  + **C1A:** Barcodes could be created and assigned to products in the product database. Furthermore, they could be sent to a label printer for products to be able to be labelled.
* **C2:** The application could scan barcodes.
  + **C2A:** The application could scan product barcodes and show detailed information related to the item upon scanning.

### Won’t – things that are out of the scope of the project

* **W1:** The application won’t be able to place deliveries
  + **W1A:** This must be done by the manager or associate manager; the app will have no integration or ability to place orders automatically. This is a manual task, outside of the project scope.
* **W2:** The application won’t act as a till for non-OneStop items
  + **W2A:** Items counts will not be changed based on every sale that goes through the till but changed on a regular stock count. This is mainly due to the inability to integrate the app with pre-existing proprietary OneStop software.

## Secondary Research

### Domain specific research

|  |  |  |  |
| --- | --- | --- | --- |
| What I need to know | Source of information | Primary / Secondary | Research method and justification |
| How to connect to a database in Python | *https://www.geeksforgeeks.org/how-to-connect-python-with-sql-database/* | Secondary research | Desk research, ***justification***: all data needs to be stored in a database and not local to the device the app is installed on therefore I need to know how to connect to a database that is running on a server in the local area network. |
| How to hash passwords in Python | *https://www.askpython.com/python/examples/storing-retrieving-passwords-securely* | Secondary research | Desk research, ***justification:*** passwords cannot be stored in plaintext inside the database, instead they must be hashed to provide a secure login system. |
| How to make a bar chart in Python | *https://www.geeksforgeeks.org/bar-plot-in-matplotlib/* | Secondary research | Desk research, ***justification:*** trends in the data located in the database need to be visualised. Graphs allow this to be communicated effectively. |
| How to send an email using Python | *https://dev.to/luca1iu/sending-emails-with-python-a-step-by-step-guide-97a* | Secondary research | Desk research, ***justification:*** once users have signed up and provided their emails, they can get emailed any notifications. For example, incorrect login attempts. |
| How to send a WhatsApp[[1]](#footnote-2) notification using Python | *https://www.geeksforgeeks.org/automate-whatsapp-messages-with-python-using-pywhatkit-module/* | Secondary research | Desk research, ***justification:*** OneStop uses *WhatsApp* to communicate with staff using a group chat. Leveraging this, the Stock Monitoring Assistant can send notifications to the group chat, allowing all staff to see any messages about information such as deliveries or the tasks completed by the end of the day. |
| How to use the camera with Python | *https://www.geeksforgeeks.org/python-opencv-capture-video-from-camera/* | Secondary research | Desk research, ***justification:*** whilst it is a future feature to be added, OneStop management have asked if it is possible to add barcode scanning abilities to the application. This would allow products to be added by scanning a barcode and then inputting the necessary data about it into the database. |
| Storing JSON list in MySQL | *https://blog.payara.fish/how-to-work-with-mysql-and-json-in-developing-cloud-apps* | Secondary research | Desk research, ***justification:*** tables in the database such as *suppliers* and *products* include datatypes in columns that require the storage of a list. As there is no way to do this in standard MySQL, using JSON is a potential solution to this issue. |
| How to get the count for how many rows there are in a MySQL table with python | *https://www.geeksforgeeks.org/how-to-count-the-number-of-rows-in-a-mysql-table-in-python/* | Secondary research | Desk research, ***justification:*** The homepage will display some general information upon logging into the application, including a welcome message and statistics such as the number of products and suppliers in the database. To do this the number of rows in either table need to be counted, as each row is equivalent to one product or one supplier respectively. This is a good way to have an overview of the system at a glance to the user. |
| How add graph to custom Tkinter | *https://pythonprogramming.net/how-to-embed-matplotlib-graph-tkinter-gui/* | Secondary research | Desk research, ***justification***: The data view included in the user interface will be used to show an overview of the tables and the data that they hold. This can be visualised using graphs embedded In a custom tkinter window. |
| How to delete the column of a MySQL table | *https://www.learningaboutelectronics.com/Articles/How-to-delete-a-column-of-a-MySQL-table-in-Python.php* | Secondary research | Desk research, ***justification***: I removed a column in the original supplier database that would have stored the list of products. However, I realised simply having a foreign key in the products table would create a many-to-one relationships which would enable a single supplier to have multiple products. |
| How to send an email with Python | *https://docs.python.org/3/library/email.examples.html* | Secondary research | Desk research, ***justification***: The program needs to be able to send emails upon user registration and when tasks are completed. This means that I need to have email functionality, set up with a new Gmail account which Python can logon to and send the necessary emails from. |
| How to update the value in a MySQL table | *https://www.w3schools.com/sql/sql\_update.asp* | Secondary research | Desk research, ***justification***: Instead of overwriting a value in a MySQL table to change it, the value can be updated instead (e.g. such as increasing the stock level) and this important to maintain data integrity. |
| How to embed a matplotlib graph in a Tkinter window | *https://www.geeksforgeeks.org/how-to-embed-matplotlib-charts-in-tkinter-gui/* | Secondary research | Desk research, ***justification:*** The data view included in the user interface will be used to show an overview of the tables and the data that they hold. This can be visualised using graphs embedded in a custom tkinter window. |
| How to iteratively create a label in tkinter | *https://stackoverflow.com/questions/60498985/how-do-i-make-a-new-label-every-time-in-a-loop-in-tkinter* | Secondary research | Desk research, ***justification:*** For the environment variables to be displayed in the settings UI, a label needs to be created for each variable in the environment file. Instead of hardcoding each label, they are created iteratively to future proof the program if more environment variables are created. |
| How to get all fields in a MySQL column python | *https://www.w3schools.com/python/python\_mysql\_select.asp* | Secondary research | Desk research, ***justification:*** This method will be used when querying items such as getting all product names from the product table, in order to be used for analysis in the weekly report algorithm. |
| Multi-threading in python | *https://www.geeksforgeeks.org/multithreading-python-set-1/* | Secondary research | Desk research, ***justification:*** Multi-threading will allow me to run 2 processes at the same time, meaning the program wont have to wait for the first process to finish before starting the second one. This is done by running a process on another thread of the CPU (if available- however, most modern processes have at least 2 threads) |

### Programming Language Research

Whilst I have been learning the high-level language *Python* steadily through Computer Science A-Level, there are also other alternatives which could offer better performance in comparison. Through researching different methods of developing cross-platform applications I came across several options:

Table 1 - (Jet Brains, 2024), (Programmingempire, 2024), (O'REILLY, n.d.), (Geeks for Geeks, 2023), (Yang bocheng, 2023)

|  |  |  |
| --- | --- | --- |
| *Language/framework* | *Advantages* | *Disadvantages* |
| Kotlin | * Mature community * Developers can reuse code across Android, iOS, web and desktop. * Easy to integrate into projects | * Can be hard to learn. * Still need Java to finish projects. * Slower in performance * Learning material is scarce as it is a relatively new language |
| Java | * Platform independent * Supports multi-threading. * Big community support * Extensive libraries & frameworks | * Java’s virtualisation overhead can impact performance. * Memory management is slow. * Syntax verbosity. |
| React Native | * Focus on the User Interface * Compatible with third-party plugins * Quick build and testing of applications on platforms * Resource optimisation | * Lower performance than native apps while handling intensive tasks. * Inefficient for complex interfaces or animations * Debugging and troubleshooting |
| .NET MAUI | * Provides cross platform APIs for accessing native device features. * Single project system * Mature ecosystem * Limitations in customizing the UI | * Steep learning curve * Stability concerns * Community is still growing with less support available compared to other options |
| Python & Tkinter | * Ease of use * Simple to use and multiple geometry management tools for GUI’s * Cross-platform compatibility * Language maturity | * Speed * UI cumbersome to design with complex interfaces * High memory consumption * Mobile application development |

#### My decision

Initially I chose to go with *Kotlin* paired with Android Studio after considering all options. While initially pleased with this combination, I found Kotlin to have a steep learning curve which was eating into much of the time I could be using to develop the project. Therefore, I substituted it with *Java*, a language which I was already familiar with. Whilst the drag and drop nature of the UI design was incredibly helpful, I ran into an issue attempting to use the *MySQL* driver and adding it to the project. Researching the issue, I found multiple solutions all of which after testing did not work. Whilst I could have spent more time finding a working solution, this was eating into my time. Therefore, I took the decision to use *Python* instead and restart the programming aspect of the development from scratch. This process deemed far quicker as I was already very used to the syntax, both through using it inside and outside of school. This meant the implementation of core features incredibly faster, and reduced the time that may have needed to be spent on researching the language’s syntax. The downside of python whilst present, after testing, I have concluded that the speed of the application matches the requirements and use case of my client.

#### Structured Query Language (SQL) – Database design

The application requires the usage of a database to connect to the information concerning all the orders, products and suppliers that the store may wish to communicate with and store information on. This will be achieved with MySQL, a widely used database definition, manipulation and transaction control language.

### Python library/package usage table of references

*\*This does not include built-in python libraries, only ones installed via PIP[[2]](#footnote-3)*

|  |  |  |
| --- | --- | --- |
| *Library/package* | *Source code* | *Purpose* |
| *customTkinter* | (Tom Schimansky, 2024) | *“Custom Tkinter is a UI-library based on Tkinter, providing modern widgets compared to the outdated look of Tkinter. Widgets and windows adapt to the system appearance and the library is compatible over Windows, Mac and Linux.”* |
| *hashlib* | (Python - Open Source, 2025) | *“A python module that implements an interface to common hashing algorithms included in FIPS as well as MD5.”* |
| *json* | (Python - Open source, 2025) | *“A lightweight data interchange format inspired by JavaScript object literal syntax (although it is not a strict subset of JavaScript”* |
| *time* | (Python - Open source, 2025) | *“This module provides various time-related functions”* |
| *smtplib* | (Python - Open source, 2025) | *“The smtplib module defines an SMTP client session object that can be used to send mail to any internet machine with an SMTP or ESMTP listener daemon.”* |
| *dotenv* | (PYPI, 2024) | *“Python-dotenv reads key-value pairs from a .env file and can set them as environment variables. It helps in the development of applications following the 12-factor principles.”* |
| *mysql.connector* | (PYPI, 2024) | *“MySQL Connector/Python enables Python programs to access MySQL databases”* |
| *matplotlib* | (Matplotlib, 2024) | *“Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.”* |
| *numpy* | (Numpy, 2024) | *“NumPy is an open-source project that enables numerical computing with Python.”* |
| *CTkPieChart[[3]](#footnote-4)* | (Akascape, 2024) | *“Another piece in the puzzle, pie chart widget for customtkinter, simple and easy to visualize any data in pie chart form.”* |
| *CTkTable* | (Akascape, 2024) | *“Simple table widget having all the basic features”* |
| *CTkXYFrame[[4]](#footnote-5)* | (Akascape, 2024) | *“A better scrollable frame for customtkinter!”* |
| *CTkDatePicker[[5]](#footnote-6)* | (Max Verwiebe, 2024) | *“CTkDatePicker is a custom date picker widget built using the CustomTkinter library. It provides a user-friendly interface for selecting dates, with both a text entry and a calendar popup for easy date selection.”* |

### Similar Systems Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *System name* | *System/hardware requirements* | *Cost* | *Ease of use* | *Purpose/functions* | *Reviews* |
| Inventoria | Windows 7/8/8.1/10/11  Works on macOS 10.5 and above | £76.54/year | “Inventoria is designed to be as intuitive to use as possible, so after a quick installation, you’ll be streamlining your inventory processes within minutes.” | Create and send purchase orders to vendors,  Track inventory levels, costs, and product history,  Set alerts for low stock and update quantities when restocking ,  Import inventory via .csv and scan barcodes for new items ,  Organize stock with categories, notes and images,  Manage inventory across multiple locations and businesses &  Integrate with other business software for seamless tracking | **“**PROS  Great for tracking inventory when receiving. Also it had purchase order forms that could be linked and customized with the company information. Data entry was very easy by filling out the basic information, you could go ahead and take advantage of helpful features, like tracking minimum stock.  CONS  This software is great having it offline, but whenever I wanted to connect so that the database could upload to the cloud or data management online, it failed. For the 3 years I had this software they did 1 update.” - (Horacio, 2018) |
| InvenTree | “The InvenTree documentation *assumes* that the operating system is a Debian based Linux OS, InvenTree requires a minimum Python version of 3.9” - (InvenTree, 2024) | Free | It is designed to be lightweight and easy to use for SME or hobbyist applications. Powerful business logic works in the background to ensure that stock tracking history is maintained, and users have ready access to stock level information. InvenTree is designed to allow for a flexible installation. | Open-source inventory management for small businesses and hobbyists, built with Python & Django,  Stock tracking, supplier management, and part organization in structured categories,  BOM management, build monitoring, and custom report generation,  API integration and plugin support for customization. | “I’ve been using InvenTree for a while, and while it’s great for stock tracking, there are a few areas that could be more user-friendly.  Navigation**:** The parts and categories system is a bit confusing. A combined navigation, like in online stores, would make it much easier to use.  Part Management**:** Adding new parts is complicated, especially in purchase orders. Simplifying this process by removing some of the stricter requirements would help non-technical staff.  Financial Reporting**:** The current financial features are limited. A more user-friendly custom report builder would make it easier to track build costs and part usage.  InvenTree is a solid tool but could be much more accessible with a few tweaks for everyday users. - (ReallyNotAVeryClever, 2023) |
| SubVentory | Windows 10 64-bit, GTX 1050, i3-8300, 8gb RAM, minimum 1gb available storage - (GameLoop, 2023) | Free |  | Save average 2 to 5 hours of labour weekly,  Helps avoid common mistakes,  More accurate inventory and cost of goods reports,  Simplified inventory process,  Easy import of invoices,  Suggestive Ordering,  Notifications from your DC - (ZippyZum, n.d.) | “A temperamental piece of garbage that is completely unusable on my phone. In the past, this app was extremely slow and buggy, but manageable. Every step required a long loading screen and if I didn't baby my phone the app would crash when it went to sleep. Now it crashes constantly even with the latest update, I have spent an hour of my free time just trying to set up an order before the next shift and with no luck” - (TheDoodleDud, 2024) |

# Design section

## ER Diagram

A computer screen shot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Table name | How is it used? |
| *Users* | The application requires a logon program to a) prevent people other than staff from accessing data relevant to the operation of the business and b) to keep track of what tasks *which* staff have carried out during their shifts, allowing any issues to be brought up with the correct staff member. |
| *Suppliers* | The suppliers need to be stored in a supplier database with each of their relevant supplier names. Supplier descriptions, the products they deliver to the business and when these delivery dates are. The “*supplier\_id”* in turn serves as a foreign key for the *Product* database, meaning that they are connected. |
| *Product* | All product types will be stored in a database including information about who supplies them, the pack size, weight and a barcode (if implemented). A description will also be included to detail what the product is and an expiration date (if applicable). |
| *Waste* | Items that fall under waste must either be damaged (for example by a customer), out of date, or stolen. Waste can be updated by staff to have a unique status, either checked, pending, or unchecked. This will allow a better management of waste in the business. Waste often happens with damaged canned drinks bursting, so the implementation of a waste system will be effective in tracking this. |
| *stockLevel* | Each product will be linked to the *stockLevel* table through the *product\_id* foreign key.This record will keep track of important data about a product, including its current stock level, minimum stock level and the reorder level. Using this data, the program runs a background check of the stock level table using multithreading, to check if the current stock level is under or close to the reorder level. If it is, an email is sent, recommending a reorder. |
| *stockLevelHistory* | The stock level history table is updated every time the stock level of a product is changed. It is used to track individual changes and when they happened, so that the trend can be either plotted in a graph or analysed to predict future trends in said stock |
| *weeklyReportRecords* | The weekly report records table is used to keep track of any calculated data that is used when making a weekly report. Information stored includes the current trend in stock level, the predicted stock levels for the next week, and a calculation of the revenue, cost of goods sold and net profit. |

## Use case diagrams and use case descriptions

|  |
| --- |
|  |

|  |  |
| --- | --- |
| *Use case name* | 1. Create new user |
| Goals in context | Admin needs to register a new staff member to the app |
| Scope and level | Overall system |
| Preconditions | Both the app and database in the OneStop office are powered on and running |
| Success end condition | New user enters their credentials, and the account is established without error in the database |
| Failed end condition | User inputs incorrect details into the signup field and therefore cannot create a new account |
| Primary actor | User with an access level of 1 (admin privileges) |
| Trigger | User chooses to create a new account |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | App asks for your personal information (including forename and email address) | | 2 | User enters their username, age, password and email address before clicking confirm | | 3 | App shows a popup notification in the middle of the screen if the profile is generated correctly and without any issues. User access level is generated automatically if it is present in the database, or the administrator selects the user access level once the new staff member has registered. | | 2.a | User enters their email address in an incorrect format. (Throws error if wrong format) | | 2.b | The user is prompted to choose a default theme or house style for the app so that when they login they have a custom experience | | 2.c | If any fields are left empty, there will be a popup asking the new user to make sure they input something into the required fields. They cannot create a new account without doing this step. | | 2.d | If the user already exists within the user database, it will prompt them to change their username or password. | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Top priority |
| Performance target | Error checking for inputs should take place and alert of any errors as soon as possible  An email should be sent that a new user has been registered, giving their details and at what time to the administrator. |
| Frequency | Each user only needs to be created once |
| Subordinate Use-Cases | Collect User information |
| Channel to primary actor | User interface, notification system |
| Secondary actors |  |
| Open issues | Should the application create a notification on the device or email it to the administrator? |
| Schedule | Second function to implement in v1.0 |
| Author | Soma Benfell 12/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Login |
| Goals in context | User inputs their login details to proceed to the homepage |
| Scope and level | Overall system |
| Preconditions | User exists in database, database in OneStop office is powered on and running |
| Success end condition | User enters their credentials and are directed to the home page. Based on user access level, different functions should appear. For example, an administrator will have access to all the tools, but a till worker might not be able to see a breakdown of profit margins |
| Failed end condition | User inputs incorrect login details into the login fields and therefore cannot proceed to the home screen |
| Primary actor | User |
| Trigger | User presses the login button under the login input fields |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | App asks for login details | | 2 | User enters correct login details into the respective fields before clicking on the login button | | 3 | App shows a “logged in successfully” notification before redirecting the user to the home page | | 2.a | User enters incorrect login details and cannot proceed to the home screen | | 2.b | User account does not exist and so cannot login to the system | | 2.c | If any fields are left empty, there will be a popup asking the user to make sure they input something into the required fields. They cannot login without doing this step | | 2.d | Database is not connected and will show an error message | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Top priority |
| Performance target | The user should be able to login to the system in under 5 seconds |
| Frequency | A login will happen every time a user needs to use one of the tools. This will depend on when there are deliveries and stock counts need to be carried out. |
| Subordinate Use-Cases | Track how often staff carry out tasks |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | n/a |
| Schedule | 1st function to implement in v1.0 |
| Author | Soma Benfell 12/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Forgot password |
| Goals in context | User clicks on the forgot password button to enter their recovery code to reset their password |
| Scope and level | Overall system |
| Preconditions | The user account already exists, and the user can remember their recovery code |
| Success end condition | Password is reset correctly |
| Failed end condition | Password cannot be reset |
| Primary actor | User |
| Trigger | User presses the forgot password button on the change password page |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | User has forgotten password and so clicks on the forgot password button | | 2 | User enters their username and recovery code | | 3 | User correctly resets their password using the temporary password generated by a valid recovery key | | 2.a | User enters incorrect recovery code and cannot reset their password | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Medium priority |
| Performance target | The user should be able to make this change in under 10s |
| Frequency | Every time/occasion on which the user forgets their password |
| Subordinate Use-Cases | n/a |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | n/a |
| Schedule | 3rd function to implement in v1.0 |
| Author | Soma Benfell 16/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Record a delivery |
| Goals in context | User inputs a delivery date, the products delivered and if there were any missing items compared to the expected list of a products. A confirmation button pops up after they click “record delivery”, and once it is confirmed, item stock levels are updated in the database accordingly. |
| Scope and level | Overall system |
| Preconditions | The database is up and running correctly, the user is logged on and there are existing products that can be added to the delivery record to increase the stock level. |
| Success end condition | Products entered in the delivery are updated accordingly in the database and the delivery is logged in recent deliveries, stating who recorded it, when, and what products were delivered. |
| Failed end condition | Delivery is unable to be recorded by the application. |
| Primary actor | User |
| Trigger | User logs in successfully, and after being redirected to the home page, clicks on the “*Record a delivery*” button under the “*tools*” section of the page. |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | User logs in and clicks on the “*Record delivery*” button | | 2 | User is taken to the “*Record a delivery*” page | | 3 | User inputs all correct information into the fields required by the page | | 4 | All information is in the correct format and delivery is submitted | | 4.a | Once submitting the delivery, said tables are updated: *stock count* and *recent deliveries* | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Top priority |
| Performance target | Once information has been inputted into all fields, updates to the correct databases should take no less than 10 seconds. |
| Frequency | Every time a delivery occurs of goods that are not supplied by OneStop. |
| Subordinate Use-Cases | n/a |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | n/a |
| Schedule | 4th function to implement in v.10 |
| Author | Soma Benfell 16/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Stock Counting |
| Goals in context | User goes around the store and inputting the count of stock for a specific product into a list shown on the app. This stock count must include stock found outside in the stock room also, not just on the shelves. Once the stock count is finished, the difference between the recorded data and the count in the database is computed, and then the database is updated accordingly. |
| Scope and level | Overall system |
| Preconditions | Database is setup and running, user is logged in and there are existing products within the database. |
| Success end condition | Product counts are entered correctly, and the database is updated with the correct count of products |
| Failed end condition | Database is unable to be updated |
| Primary actor | User |
| Trigger | User logs into the application and clicks on the “S*tock counting*” option under the “*tools*” section |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | User logs in and clicks on the “*Stock counting*” button | | 2 | User is taken to the “*Stock counting*” page | | 3 | User chooses a product and selects the count | | 4 | User confirms stock counts | | 2.a | The database connection is tested to make sure that it is online, if not, a warning is issued | | 3.a | The user can increase or decrease the count if a wrong number was entered – validation is used to ensure that this a number in the range of 1 to a realistic amount (no more than what is already in the stock database) | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Top priority |
| Performance target | Depending on how many products are counted, the updates to the database should take no longer than 10 seconds. |
| Frequency | Depending on when stock counts are carried out for products, this could be once or twice every few weeks. |
| Subordinate Use-Cases | n/a |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | n/a |
| Schedule | 5th function to implement in v1.0 |
| Author | Soma Benfell 17/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Data view |
| Goals in context | Provide a view of the data stored in the databases for: products, stock counts, suppliers, recent deliveries and any waste |
| Scope and level | Overall system |
| Preconditions | User is logged into the application and the database is online and running. Whilst it is not necessary for any information to be in there, nothing will display unless products and suppliers are added. |
| Success end condition | Data from the database is displayed accordingly |
| Failed end condition | Database cannot be connected to and so the app displays an error message |
| Primary actor | User |
| Trigger | User logs in and selects the “*Data view*” button under the “*Tools*” section |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | User logs into the application | | 2 | User can navigate to the data view screen by selecting the “*Data view*” button. | | 3 | User can select from a tabulated window which data they wish to view | | 2.a | Data is displayed depending on user access level. This is considered when the *data-view* button onclick event is called, where the user is directed to a different page depending on their information in the database. | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Top priority |
| Performance target | The application should take no longer than 20 seconds to display this data. This could vary depending on which data is being loaded. Not all tables will be loaded simultaneously, rather when which the tab under which that table falls under is clicked on. This is done to reduce resource use and speed up the application. |
| Frequency | Every time a staff member wishes to view data. |
| Subordinate Use-Cases | n/a |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | n/a |
| Schedule | 6th function to implement in v1.0 |
| Author | Soma Benfell 17/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Settings |
| Goals in context | The button to go to the settings page should be visible on almost all pages of the application, allowing it to be easy to access. It should have important settings, such as which database to connect to, resetting your password and choosing a theme to tailor a user’s experience in the app. |
| Scope and level | Overall system |
| Preconditions | User is logged in and the device is connected to the internet |
| Success end condition | User is able to change settings, and any updates should be reflected across their profile |
| Failed end condition | User cannot navigate the settings panel or setting are not able to be updated/changed |
| Primary actor | User |
| Trigger | User presses on the settings button |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | User presses on the settings button | | 2 | User changes a setting – such as creating a user or an environment variable | | 3 | Application updates any settings | | 2.a | Application checks if user has the correct access level to edit said setting | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Medium priority |
| Performance target | Settings should be able to be updated in under 10 seconds |
| Frequency | Every time a setting needs to be changed |
| Subordinate Use-Cases | n/a |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | Settings button crashes app and take you back to the home page |
| Schedule | 7th function to implement in v1.0 |
| Author | Soma Benfell 17/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Create product |
| Goals in context | To be able to create a new product so that it can be added to the database |
| Scope and level | Overall system |
| Preconditions | User is logged in and the device is connected to the internet, a supplier needs to exist |
| Success end condition | New product is created |
| Failed end condition | New product is unable to be created |
| Primary actor | User |
| Trigger | User clicks on the button to “*add product*” |
| Success scenario  Extensions | |  |  | | --- | --- | | Step | Action | | 1 | User clicks on the button to “*add product*” | | 2 | User enters details for a new product | | 3 | New product is uploaded to the database | | 3.a | Application checks that a supplier has been selected (has to exist beforehand) | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Medium priority |
| Performance target | Product creation should take no longer than 5 seconds (for the process of uploading it to the database) |
| Frequency | Every time a new product is needed to be created. This will be more frequent when the app is initially set up at OneStop, as there is not existing data that can be integrated with the Stock Monitoring Assistant, since OneStop data is out of bounds to the project. |
| Subordinate Use-Cases | n/a |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | n/a |
| Schedule | 8th function to implement in v1.0 |
| Author | Soma Benfell 18/11/24 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| *Use case name* | 1. Create supplier |
| Goals in context | To be able to create a new supplier so that it can be added to the database |
| Scope and level | Overall system |
| Preconditions | User is logged in and the device is connected to the internet |
| Success end condition | New supplier is created |
| Failed end condition | New supplier is unable to be created |
| Primary actor | User |
| Trigger | User clicks on the button to “*add supplier*” |
| Success scenario | |  |  | | --- | --- | | Step | Action | | 1 | User clicks on the button to “*add supplier*” | | 2 | User enters details for a new supplier, including what products they supply | | 3 | New supplier is uploaded to the database | |

|  |  |
| --- | --- |
| *Related information* | |
| Priority | Medium priority |
| Performance target | Supplier creation should take no longer than 5 seconds (for the process of uploading it to the database) |
| Frequency | Every time a new supplier is needed to be created. This will be more frequent when the app is initially set up at OneStop, as there is not existing data that can be integrated with the Stock Monitoring Assistant, since OneStop data is out of bounds. |
| Subordinate Use-Cases | n/a |
| Channel to primary actor | User interface, database system and notification system. |
| Secondary actors | n/a |
| Open issues | n/a |
| Schedule | 9th function to implement in v1.0 |
| Author | Soma Benfell 18/11/24 |

## Data dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Session |  | | | |
| *Field Name* | *Type* | *Extends* | *Validation* | *Notes* |
|  |  |  |  |  |
| Users table | | | | |
| *user\_id* | Auto number |  | Int not null  Primary key | Unique identifier for each user |
| *username* | String |  | varchar(50) not null | Staff member username (could be the same as forename but allows for changes) |
| *password* | String |  | varchar(64) not null | Each password hash is 64 characters long. In the login system, the entered password is hashed and compared to the one in the table |
| *access\_level* | Int |  | Int not null | The user’s level of access to certain tools and functions within the app |
| *recovery\_code* | String |  | varchar100) not null | Generated upon user created, allows you to reset your forgotten password. (OTP) |
| *email\_address* | String |  | Character not null | Users email address to be able to send notifications such as when logging in or when a password has been entered incorrectly. |
|  | | | | |
| Product table | | | | |
| *product\_id* | Auto number |  | Int not null  Primary key | The unique identifier for a product. |
| *supplier\_id* | Auto number |  | Character not null  Foreign key | Unique identifier for each user |
| *product\_name* | String |  | varchar(100) not null | The name for a product. |
| *product\_description* | String |  | varchar(200) not null | A description of what the product is |
| *product\_pack\_size* | Int |  | Int not null | Pack size of a product, 1 if it is not a pack but a standalone item |
| *product\_weight* | Int |  | Int not null | Product weight in grams |
| *product\_barcode* | String of digits |  | varchar50) | Not to be implemented in v1.0 |
| *product\_buy\_price* | decimal(10, 2) |  | decimal(10, 2) not null | A decimal number allowing up to 10 digits in total with 2 digits after the decimal point, storing the price at which the product is ***purchased*** by *OneStop* |
| *Product\_purchase\_price* | decimal(10, 2) |  | decimal(10, 2) not null | A decimal number allowing up to 10 digits in total with 2 digits after the decimal point, storing the price at which the product is ***sold*** by *OneStop* |
|  | | | | |
| Suppliers table | | | | |
| *supplier\_id* | Auto number |  | Int not null  Primary key | Unique identifier for each supplier |
| *supplier\_name* | String |  | Character not null | Supplier name / company name |
| *supplier\_description* | String |  | Character not null | Description of the supplier |
| *supplier\_delivery\_dates* | JSON |  | n/a | The days on which a supplier delivers any products. There could be multiple delivery dates so this will be stored in a JSON list |
|  | | | | |
| Waste table | | | | |
| *waste\_id* | Auto number |  | Int not null  Primary key | The unique identifier for each item of waste |
| *product\_id* | Auto number |  | Int not null  Foreign key | Foreign key – n/a |
| *supplier\_id* | Auto number |  | Int not null  Foreign key | Foreign key – n/a |
| *waste\_reason* | String |  | varchar(200) | A short description of why the item is considered waste |
| *waste\_dealt\_with* | Boolean |  | tinyint(1) | A status of the process, to detail whether it has been dealt with or still waiting to be dealt with. |
|  | | | | |
| Stock level table | | | | |
| *stock\_id* | Auto number |  | Int not null  Primary key | The unique identifier for an item of stock |
| *product\_id* | Auto number |  | Int not null  Foreign key | Foreign key – n/a |
| *stock\_count* | Int |  | Int not null | An integer representing the amount of that product currently in stock |
| *minimum\_stock\_level* | Int |  | Int not null | A minimum level of stock that when reached should trigger a notification advising to re-order quickly, with high priority judging on the trend at which the product is selling |
| *reorder\_level* | Int |  | Int not null | A level when reached should advise a re-order |
| *last\_delivery* | JSON |  | n/a | A list value, the last time this product was delivered. JSON format for compatibility with delivery date column values from the supplier table |
|  |  |  |  |  |
| Stock level history table | | | | |
| *stockHistory\_id* | Auto number |  | Int not null  Primary key | The unique identifier for a stock history level record |
| *stock\_id* | Int |  | Int not null  Foreign key | Foreign key – n/a |
| *product\_id* | Int |  | Int not null  Foreign key | Foreign key – n/a |
| *stock\_history\_product\_name* | Varchar(100) |  | Varchar not null | The name of the product that the stock level history is being created for. |
| *stock\_count* | Int |  | Int not null | The stock count at the time that the record was created. This happens whenever there is an update to the *stock level* table. |
| *action* | Varchar(100) |  | Varchar not null | Used to store the type of stock level history record, “delivery” if the updated stock level was from a delivery and “count” if it originated from a stock count performed by a member of staff |
| *date* | Datetime |  | not null default current\_timestamp | The date and time at which the stock level history record is created so that it can be plotted on a graph against the stock count to track changes in stock level over time |
|  |  |  |  |  |
| Weekly reports record table | | | | |
| *weekly\_record\_report\_id* | Auto number |  | Int not null  Primary key | The unique identifier for a weekly reports record |
| *product\_id* | Int |  | Int not null  Foreign key | Foreign key – n/a |
| *weekly\_report\_generation\_date* | Datetime |  | not null default current\_timestamp | The date and time at which the weekly report record is created. This will allow records to be grouped by date for when the user selects a previous weekly report to view. |
| *trend* | Varchar(100) |  | Varchar not null | Stores the analysed trend of the previous week of stock level changes for a product, using linear regression |
| *predicted* | JSON |  | n/a | Stores a list of predicted stock levels for the upcoming week based on data from the previous week. More accurate if there’s more data to analyse. |
| *revenue* | Decimal(10, 2) |  | decimal(10, 2) not null | Used to store the calculated revenue to a prevision of 2 decimal places |
| *cost\_of\_goods\_sold* | Decimal(10, 2) |  | decimal(10, 2) not null | Used to store the calculated cost of goods sold to a prevision of 2 decimal places |
| *net\_profit* | Decimal(10, 2) |  | decimal(10, 2) not null | Used to store the calculated net profit to a prevision of 2 decimal places |

## UML class diagram

### DB Handling classes

A diagram of a computer

AI-generated content may be incorrect.

### Process classes

A diagram of a computer program

Description automatically generated with medium confidence

### Main.py classes

|  |  |
| --- | --- |
| A screen shot of a computer code  AI-generated content may be incorrect. | A screen shot of a computer code  AI-generated content may be incorrect. |

## Initial UI wireframe diagram (Kotlin/Java – prior lang. change)

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Application template | Logon window | Menu options |
|  |  |  |
| Recording a delivery | Stock counting | Data view |

## Final UI wireframe diagram (Python – after lang. change)

### Discussion and changes

Due to the nature of *Python,* a user interface similar to the one outlined in the initial design, is not necessarily achievable. Whilst there are certain libraries that can work on mobile devices in a fashion similar to that of *Java/Kotlin* paired with *Android Studio*, the functionality is not the same, especially when constructing more complex interfaces. For this reason, I chose a window-based approach, alike to the interfaces one might find on a Windows OS machine. Whilst this means that staff would have to use a device running windows, the functionality is maintained. It also improves cross platform compatibility, meaning as long as you are connected to the Local Area Network (LAN), anyone logged onto the app can view the data and perform tasks. Whilst a phone layout may be ideal for a staff member carrying out stock control, a window-based approach allows for more screen real estate, such as when an admin is accessing data from a tower computer in the office adjacent to the shop floor. This greatly increases the user experience and allows any processes to be carried out effectively, as there is space for the UI to include more complex layouts, something that would have to be “dumbed down” on a mobile device due to restrictions in screen space and still trying to maintain an application that can be, at face value, easily understood by any staff member.

### Updated wireframe designs

|  |  |
| --- | --- |
| *Login window* | *Menu window* |
| A screenshot of a computer login  Description automatically generated | A screenshot of a computer  Description automatically generated |
| The login window takes the usual username and password as any other application but adds forgot password functionality on the same window for ease of use in case a staff member forgets their password. This works on the basis of a recovery key, which is generated for each user accounts upon its creation. This key is regenerated after every password change. If the username and password are correct, the login window is destroyed, and the user is directed to the home page of the application. | This is a general menu window UI diagram. The tabulated view on the right contains all the pages that are needed for the program, which is navigable by the large vertical navigation bar on the left-hand side of the application. Additionally, the tabulated view is also navigable by the tab names on the top of the frame, which serve as both pointers for which tab you are on and buttons for going to a different page in the application. A simple logo is displayed in the top left-hand side of the vertical navigation bar. |
| *Home page* | *Record delivery* |
| A screenshot of a computer  Description automatically generated | A screenshot of a computer  AI-generated content may be incorrect. |
| The home page displays a greeting message to the user based on their username. Below this, there is a breakdown of the tables in the database and the percentage of rows they own out of total rows across the whole database. This is a clean way to display information about the application. A key is placed on the right hand side of this pie chart, showing which coloured section of the pie chart is showing which table. In testing, when the user isnt logged in, the welcome message defaults to “Hi user!”. | The record delivery UI allows the user to select a supplier and a delivery date. The delivery date calendar picker automatically assumes the current day is the day of the delivery, however this can be changed. A user can search for a product, enter a quantity and add it to the delivery. The space below the quantity entry is scrolling to accommodate multiple product entries in the delivery. Once as product is added to the delivery, it is added to the list of items in the delivery. The user can then change the quantity of the product as they so wish (for example if they entered the wrong amount without realising), or they want to remove the product they can press the delete button next to each item. |
| *Data view* | *Weekly report* |
| A screenshot of a computer  Description automatically generated | A screenshot of a computer  AI-generated content may be incorrect. |
| The data view tab is used to display data in the important tables. This includes products, suppliers, stock level and waste items. The data view UI consists of an additional tabulated view for each table present in the aforementioned tables. Each tab consists of a search function and the table itself. The search function should consist of an autocomplete as to make usability for staff easier. Once the search button is pressed, the found record is highlighted in the table, and any respective functions for that table popup. This includes visualising the stock level for products in the form of a line graph and resolving waste statuses for waste products. | The weekly report is separated into two sections. The top has a drop-down box containing previous weekly reports, and a “see previous report” button that shows what products and their associated calculations (such as profit margins) were calculated for that selected period. The button simply displays all this data in a legible manner inside the scrolling frame. Below the separator, a user can generate a weekly report. Optionally they can choose to also email this (to the default email noted in the environment variables), and they can choose to produce a .txt output, which can be printed off. These files are stored in the same directory as the program itself. |

# Technical implementation

## Resources

#### Technical requirements

|  |  |  |  |
| --- | --- | --- | --- |
| *Device* | *Technical specifications* | *Dependencies* | *Parts run* |
| PC | **CPU**: Intel i3 10100f @ 3.6Ghz, 4-cores, 8-threads  **RAM**: 16gb Corsair Vengeance DDR4 @ 2666Mhz  **GPU**: AMD Radeon Sapphire Nitro rx570 8gb OC  **Storage**: 256gb NVME  **OS**: Windows 11 | Visual studio code  MySQL Community server 9.2.0 Innovation  – (https://dev.mysql.com/downloads/mysql/)  MySQL Workbench **8.0.41**  - (https://dev.mysql.com/downloads/workbench/) | MySQL Database |
| MacBook air | **CPU**: Apple M3, 8-cores  **RAM**: 24gb unified memory  **GPU:** 8-core, hardware-accelerated ray tracing  **Storage**: 512gb SSD  **OS**: MacOS Sequoia 15.3.1 | Visual studio code  MySQL Community server 9.2.0-arm64 Innovation  – (https://dev.mysql.com/downloads/mysql/)  MySQL Workbench **8.0.41**  **-** (https://dev.mysql.com/downloads/workbench/) | MySQL Database |
| Raspberry Pi 4b | **CPU**: (ARM v8) 64-bit SoC @ 1.8GHz  **RAM**: 4GB LPDDR4-3200 SDRAM  **GPU**: Broadcom Videocore VI  **Storage**:16gb SD-Card  **OS**: Raspberry Pi OS (Debian) | MySQL Community server 9.2.0 Innovation  – (https://dev.mysql.com/downloads/mysql/) | MySQL Database |

As seen in the table above, all three computers are running the *MySQL community server* and *MySQL* *Workbench*. This is because, to develop the project at home, I would mainly use my Windows PC, due to its processing power and the fact that it was connected to a larger external monitor. This made the development process much easier. I used the database on my *MacBook air* so that I could program on the go, such as when I had free study periods. Finally, the Raspberry Pi database was used when we were in lesson time, in computer science, and were programming in an IT room. I would often use my *MacBook* to connect to the *RPI* on the local network.

## Code

### Project structure contents page

Python Logo PNG Transparent Images<Python File> mainApp.py \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<Place in this file> [[init and lib imports](#initMethod)]

<Place in this file> [[setButtonStates](#setButtonStates)]

<Place in this file> [[homeUI](#homeUI)]

<Place in this file> [[recordDeliveryUI](#recordDeliveryUI)]

<Place in this file> [[recordDeliveryUI functionality](#recordDeliveryUIFunctionality)]

<Place in this file> [[stockCountingUI](#stockCountingUI)]

<Place in this file> [[stockCountingUI functionality](#stockCountingUIFunctionality)]

<Place in this file> [[dataViewUI](#dataViewUI)]

<Place in this file> [[dataViewUI functionality](#dataViewUIFunctionality)]

<Place in this file> [[addProductUI](#addProductUI)]

<Place in this file> [[addProductUI functionality](#addProductUIFunctionality)]

<Place in this while> [[addSupplierUI](#addSuppplierUI)]

<Place in this file> [[addSupplierUI functionality](#addProductUIFunctionality)]

<Place in this file> [[wasteUI](#wasteUI)]

<Place in this file> [[wasteUI functionality](#wasteUIFunctionality)]

<Place in this file> [[weeklyReportUI](#weeklyReportUI)]

<Place in this files> [[weeklyReportUI functionality](#weeklyReportUIFunctionality)]

<Place in this file> [[settingsUI](#settingsUI) & functionality]

<Place in this file> [[uiWidgetClearer](#uiWidgetClearer)]

<Place in this file> [[isRowEmpty](#isRowEmpty)]

<Place in this file> [[if \_\_name\_\_ == “\_\_main\_\_”](#mainFileRunning)]

📁 *<folder> config \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<ENV File> .env [[environment variables]](#environmentVariables)

<.txt File> staticSetUpDate.txt [[program set up state]](#programSetUpStatus)

📁 *<folder> dbHandling \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<Python File> \_\_init\_\_.py [empty file, simply for allowing package-style importing]

<Python File> DBhandler.py [[dbHandler](#dbHandler)]

<Python File> logonDBHandler.py [[logonDBHandler](#logonDBHandler)]

<Python File> productDBHandler.py [[productDBHandler](#productDBHandler)]

<Python File> stockLevelDBHandler.py [[stockLevelDBHandler]](#stockLevelDBHandler)

<Python File> stockLevelHistoryDBhandler.py [[stockLevelHistoryDBHandler]](#stockLevelHistoryDBHandler)

<Python File> supplierDBHandler.py [[supplierDBHandler]](#supplierDBHandler)

<Python File> wasteDBHandler.py [[wasteDBHandler]](#wasteDBHandler)

<Python File> weeklyReportRecordDBHander.py [[weeklyReportRecordDBHander](#weeklyReportRecordDBHandler)]

📁 *<folder> icos \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<.ico File> person.ico [[person](#person)]

📁 *<folder> processes \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<Python File> \_\_init\_\_.py [empty file, simply for allowing package-style importing]

<Python File> autoCompleteSearch.py [[autoCompleteSearch](#autoCompleteSearch)]

<Python File> changePassword.py [[changePassoword]](#changePassword)

<Python File> checkSetUp.py [empty file, an intended future addon to the application]

<Python File> CTkDatePicker.py [[CTkDatePicker.py]](#CTkDatePicker)

<Python File> doubleAxesScrollingFrame.py [[CTkXYFrame.py]](#doubleAxesScrollingFrame)

<Python File> forgotPassword.py [[forgotPassword.py]](#forgotPassword)

<Python File> loginProcess.py [[loginProcess.py]](#loginProcess)

<Python File> newUser.py [[newUser.py]](#newUser)

<Python file> pieChart.py [[pieChart.py]](#pieChart)

<Python File> popUpWindow.py [[popUpWindow.py]](#popUpWindow)

<Python File> sendEmail.py [[sendEmail.py]](#sendEmail)

<Python File> stockLevelChecker.py [[stockLevelChecker.py]](#stockLevelChecker)

<Python File> windowSuperClass.py [[windowSuperClass.py]](#windowSuperClass)

📁 *<folder> templates \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<.txt File> envVarTemplate.txt [[envVarTemplate.txt]](#envVarTemplate)

📁 *<folder> weeklyReports \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<.txt File> weeklyReport\_xx-xx-xxx.txt [[weeklyReport\_xx-xx-xx.txt]](#weeklyReport)

### Source

*Requirements can be found in the* [*MoSCoW analysis*](#_MoSCoW_Analysis)

*mainApp.py source*

|  |  |  |
| --- | --- | --- |
| mainApp.py  - \_\_init\_\_ method | Requirement(s) met: **M10B**, **S5** | Technicality: Group B |
| #general lib imports  import customtkinter  from tkinter import messagebox  from time import gmtime, strftime  from datetime import timedelta  import json  import dotenv  from functools import reduce  import threading  from CTkTable import \*  import os  from os import path  import sys  import shutil  from datetime import \*  #import processes  from processes.changePassword import \*  from processes.loginProcess import \*  from processes.popUpWindow import \*  from processes.windowSuperClass import superWindow  from processes.autoCompleteSearch import AutocompleteEntry  from processes.newUser import \*  from processes.stockLevelChecker import \*  #not programmed by me  from processes.pieChart import \*  from processes.doubleAxesScrollingFrame import \*  from processes.CTkDatePicker import \*  #import database handlers  from dbHandling.logonDBHandler import \*  from dbHandling.productDBHandler import \*  from dbHandling.supplierDBHandler import \*  from dbHandling.wasteDBHandler import \*  from dbHandling.stockLevelDBHandler import \*  from dbHandling.stockLevelHistoryDBHandler import \*  from dbHandling.weeklyReportDBHandler import \*  #main app class  class App(superWindow):  WIDTH = 1100  HEIGHT = 675  \_appLocation = os.path.dirname(os.path.abspath(sys.argv[0]))  \_defAlertEmail = os.getenv('DEF\_ALERT\_RECIPIENT\_EMAIL')  def \_\_init\_\_(self, userAccessLevel, userName="user"):  super().\_\_init\_\_()  self.userAccessLevel = userAccessLevel  self.userName = userName  #configure window  self.title("OneStop Stock Assistant System")  self.geometry(f"{App.WIDTH}x{App.HEIGHT}")  self.resizable(True, False)  #configure grid layout (4x4)  self.grid\_columnconfigure(1, weight=1)  self.grid\_rowconfigure((0, 1, 2), weight=1)  #create sidebar frame with widgets and create the logo text  self.sidebarFrame = customtkinter.CTkFrame(self, width=140, corner\_radius=0)  self.sidebarFrame.grid(row=0, column=0, rowspan=4, sticky="nsew")  self.sidebarFrame.grid\_rowconfigure(4, weight=1)  self.logoLabel = customtkinter.CTkLabel(self.sidebarFrame, text="OSSMA", font=customtkinter.CTkFont(size=20, weight="bold"))  self.logoLabel.grid(row=0, column=0, padx=20, pady=(20, 10))  #create a section of buttons for stock taking tools  self.label1 = customtkinter.CTkLabel(self.sidebarFrame, text="Stock taking tools:", font=customtkinter.CTkFont(size=12))  self.label1.grid(row=1, column=0, padx=20)  self.sideBarButton1 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Record a delivery"), text="Record a delivery")  self.sideBarButton1.grid(row=2, column=0, padx=20, pady=10)  self.sideBarButton2 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Stock counting"), text="Stock counting")  self.sideBarButton2.grid(row=3, column=0, padx=20, pady=10)  #exit button  self.sidebarExitButton = customtkinter.CTkButton(self.sidebarFrame, command=self.onClosing, text="Exit")  self.sidebarExitButton.grid(row=4, column=0, padx=20, pady=10)    #create a section of buttons for database tools, such as adding a product or supplier  self.label2 = customtkinter.CTkLabel(self.sidebarFrame, text="Database tools:", font=customtkinter.CTkFont(size=12))  self.label2.grid(row=5, column=0, padx=20)  self.sideBarButton3 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Data view"), text="Data view")  self.sideBarButton3.grid(row=6, column=0, padx=20, pady=10)  self.sideBarButton4 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Add product"), text="Add product")  self.sideBarButton4.grid(row=7, column=0, padx=20, pady=10)  self.sideBarButton5 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Add supplier"), text="Add supplier")  self.sideBarButton5.grid(row=8, column=0, padx=20, pady=10)  self.sideBarButton6 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Waste"), text="Waste")  self.sideBarButton6.grid(row=9, column=0, padx=20, pady=10)  #create a section of buttons for tools that present data in graph format etc  seperator2 = customtkinter.CTkFrame(self.sidebarFrame, height=1, width=100,fg\_color="gray")  seperator2.grid(row=10, column=0, padx=20, pady=10)  self.label3 = customtkinter.CTkLabel(self.sidebarFrame, text="Data tools:", font=customtkinter.CTkFont(size=12))  self.label3.grid(row=11, column=0, padx=20)  self.sideBarButton7 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Weekly report"), text="Weekly report")  self.sideBarButton7.grid(row=12, column=0, padx=20, pady=10)  self.sideBarButton8 = customtkinter.CTkButton(self.sidebarFrame, command=lambda: self.goToTab("Settings"), text="Settings")  self.sideBarButton8.grid(row=13, column=0, padx=20, pady=(10,20))  #========================BUTTON-STATES=======================>  #tabview in which all UI will take place to do with functions of the application - the sidebar on the side simply allows for easier switching of the tabs  self.setButtonStates() #set the button states (disabled or enabled) based on the user access    #create the tabview according to allowances  self.tabview = customtkinter.CTkTabview(master=self)  self.tabview.grid(column=1, row=0)  for tab in self.tabsDefault:  self.tabview.add(tab)  #disable any buttons & tabview tabs that the user doesnt have access to  for page in self.tabsDefault:  if page not in self.allowances[int(self.userAccessLevel)]:  self.tabview.delete(page)  for button in self.buttonsDefault:  name = button.cget("text")  if name == page:  button.configure(state="disabled")    #<========================INITIALIZE-DATABASES========================>  self.DBHandler = DBHandler()  self.supplierDB = supplierDBHandler()  self.productDB = productDBHandler()  self.wasteDB = wasteDBHandler()  self.stockLevelDB = stockLevelDBHandler()  self.stockLevelHistoryDB = stockLevelHistoryDBHandler()  self.weeklyReportDB = weeklyReportDBHandler()  databases = [self.supplierDB, self.productDB, self.wasteDB, self.stockLevelDB, self.stockLevelHistoryDB, self.weeklyReportDB]  for database in databases:  try:  database.initializeDatabase()  except Exception as error:  print(error)  #<========================UI-SETTERS========================>  uiSetters = [  [self.homeUI, 3],  [self.recordDeliveryUI, 3],  [self.stockCountingUI, 3],  [self.dataViewUI, 1],  [self.addProductUI, 2],  [self.addSupplierUI, 2],  [self.wasteUI, 2],  [self.weeklyReportUI, 1],  [self.settingsUI, 1]  ]    for uiSetter in uiSetters:  if int(self.userAccessLevel) <= uiSetter[1]:  uiSetter[0]() | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py - setButtonStates method | Requirement(s) met: **M10B** | Technicality: Group A |
| #function for buttons in the sidebar - used for navigating the tabview on the right      def goToTab(self, tabName):          self.tabview.set(tabName)      def setButtonStates(self):          #get user access level from login program in order to disable some functions          self.tabsDefault = ["Home", "Record a delivery", "Stock counting", "Data view", "Add product", "Add supplier", "Waste", "Weekly report", "Settings"]          self.buttonsDefault = [self.sideBarButton1, self.sideBarButton2, self.sideBarButton3, self.sideBarButton4, self.sideBarButton5, self.sideBarButton6, self.sideBarButton7, self.sideBarButton8]          self.tabs = self.tabsDefault          self.allowances: dict = {                  1: self.tabsDefault,                  2: list(filter(lambda tab\_: tab\_ not in ["Data view", "Weekly report", "Settings"], self.tabs)),                  3: list(filter(lambda tab\_: tab\_ not in ["Add product", "Add supplier", "Data view", "Weekly report", "Settings", "Waste"], self.tabs))          } | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – homeUI method | Requirement(s) met: **M11, M11A** | Technicality: Group B |
| def homeUI(self, tab\_='Home'):          self.tab\_ = tab\_          #Configure the grid to center for the centered welcome text          self.tabview.tab(tab\_).grid\_rowconfigure([0, 1, 2], weight=1)          self.tabview.tab(tab\_).grid\_columnconfigure([0, 1, 2, 3], weight=1)          #set the welcome label in the center          self.welcomeLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text=f"Hi {self.userName}!", font=customtkinter.CTkFont(size=35, weight="bold"), padx=0, pady=0)          self.welcomeLabel.grid(row=0, column=0, columnspan=4, pady=(30,30))          self.databaseBreakdown = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Database table breakdown:", font=customtkinter.CTkFont(size=15, weight="normal"), padx=0, pady=0)          self.databaseBreakdown.grid(row=1, column=0, columnspan=4, pady=(15,30))          #create the pie chart for displaying the table values          pieChart = CTkPieChart(self.tabview.tab(tab\_), line\_width=50)          pieChart.grid(row=2, column=0, padx=10, pady=(10, 50), sticky="e")          try:              pieChart.add("Products", self.productDB.getCount("products", False), text\_color="black", color="#1F538D")              pieChart.add("Suppliers", self.productDB.getCount("suppliers", False), text\_color="black", color="gray")              pieChart.add("Waste", self.productDB.getCount("waste", False), text\_color="black", color="green")              pieChart.add("Stocklevel", self.productDB.getCount("stocklevel", False), text\_color="black", color="purple")              pieChart.add("Users", self.productDB.getCount("users", False), text\_color="black", color="yellow")              pieChart.add("StockLevelHistory", self.productDB.getCount("stocklevelhistory", False), text\_color="black", color="indigo")          except TypeError:              pass            #get the dictionary of key value pairs to create the key for the pie chart          pieChartValues = pieChart.get()          #create a frame for the piechart          pieChartFrame = customtkinter.CTkFrame(self.tabview.tab(tab\_), fg\_color="transparent")          pieChartFrame.grid(row=2, column=1, padx=10, pady=10, sticky="e", columnspan=2)          #display the values for the key          for key, values in pieChartValues.items():              dataCircle = customtkinter.CTkRadioButton(pieChartFrame, hover=False, text=key, width=1,fg\_color=values["color"])              dataCircle.select()              dataCircle.pack(side='top', anchor='nw', pady=5) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – recordDeliveryUI method | Requirement(s) met: **M4, M4A** | Technicality: Group B |
| def recordDeliveryUI(self, tab\_='Record a delivery'): #you might want to make this a scrollable fram          #you need to create a supplier database and then select all suppliers in order to be able to give values for the value list below          self.tab\_ = tab\_            self.chooseSupplierLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Choose supplier:", anchor="w")          self.chooseSupplierLabel.grid(row=0, column=0, padx=(20, 20), pady=20, sticky='w')          #if no data in table, CTKOptionMenu throws an error, so try except block creates failure lable if this issue is encountered          try:              self.chooseSupplier1 = customtkinter.CTkOptionMenu(self.tabview.tab(tab\_), dynamic\_resizing=False, values=self.supplierDB.getSupplierNames(), width=200) #values list should be taken from a database call once the supplier database is created              self.chooseSupplier1.grid(row=0, column=1, padx=20, pady=20)            except Exception as error:              self.noSupplierLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="No suppliers found", anchor="w")              self.noSupplierLabel.grid(row=0, column=1, padx=(20, 20), pady=20, sticky='w')          #delivery date shoud create an ovveride feature if user doesnt want t   o use current system date          self.enterDeliveryDateLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Delivery date:", anchor='w')          self.enterDeliveryDateLabel.grid(row=1, column=0, padx=(20, 20), pady=20, sticky='w')          self.deliveryDate = CTkDatePicker(self.tabview.tab(tab\_))          self.deliveryDate.grid(row=1, column=1, padx=(15, 20), pady=10, sticky='w')          self.month = str(datetime.now().month)          if len(self.month) == 1:              self.month = f"0{datetime.now().month}"          self.deliveryDate.date\_entry.insert(0, f"{datetime.now().day}/{self.month}/{datetime.now().year}")          #delivery time shoud create an ovveride feature if user doesnt want to use current system time          self.enterDeliveryTimeLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Delivery time:")          self.enterDeliveryTimeLabel.grid(row=1, column=2, padx=(20, 20), pady=20, sticky='w')          self.deliveryTime = strftime("%X", gmtime())          self.enterDeliveryTimeLabelAbs = customtkinter.CTkLabel(self.tabview.tab(tab\_), text=self.deliveryTime)          self.enterDeliveryTimeLabelAbs.grid(row=1, column=3, padx=(20, 20), pady=20, sticky='w')          #create the autocomplete search for a product          self.findProductLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Search product:")          self.findProductLabel.grid(row=3, column=0, padx=(20), pady=20, sticky='w')          self.autocompleteEntry = AutocompleteEntry(self.tabview.tab(tab\_), width=500, placeholder\_text='Search product...')            self.autocompleteEntry.setSuggestions(self.productDB.getProductNames()) #set suggestions needs to be based on a call to the product table in the database          self.autocompleteEntry.grid(row=3, column=1, padx=20, pady=20, columnspan=3, sticky='w')            self.quantityLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Quantity: ")          self.quantityLabel.grid(row=4, column=0, padx=(20, 20), pady=10, sticky='w')          self.quantityEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="x")          self.quantityEntry.grid(row=4, column=1, padx=(20, 20), pady=10, sticky='w')          self.addProduct = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Add product", command=self.addProductToDelivery)          self.addProduct.grid(row=4, column=2, padx=20, pady=10)          #create a seperator to distuinguish between sections          seperator2 = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")          seperator2.grid(row=5, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')          #scrollable frame for added products          self.products = []          self.productFrame = customtkinter.CTkScrollableFrame(master=self.tabview.tab(tab\_), width=300, height=200, corner\_radius=0, fg\_color="transparent")          self.productFrame.grid(row=6, column=0, sticky="nsew", columnspan=6)          self.productNumLabel = customtkinter.CTkLabel(self.productFrame, text="Item num", fg\_color="transparent")          self.productNumLabel.grid(row=0, column=0, padx=(20), pady=20, sticky='w')          self.itemLabel = customtkinter.CTkLabel(self.productFrame, text="Item", fg\_color="transparent")          self.itemLabel.grid(row=0, column=1, padx=(20), pady=20, sticky='w')          self.itemQuantityLabel = customtkinter.CTkLabel(self.productFrame, text="Quantity", fg\_color="transparent")          self.itemQuantityLabel.grid(row=0, column=2, padx=(20), pady=20, sticky='w')          self.toolLabel = customtkinter.CTkLabel(self.productFrame, text="Tool")          self.toolLabel.grid(row=0, column=3, padx=(20), pady=20, sticky='w')            #create a seperator to distuinguish between sections          seperator3 = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")          seperator3.grid(row=7, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')          self.confirmDelivery = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Confirm delivery", command=self.confirmDelivery)          self.confirmDelivery.grid(row=8, column=0, padx=20, pady=10) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – recordDeliveryUI functionality | Requirement(s) met: **M4, M4B** | Technicality: Group B |
| #function to add product      def addProductToDelivery(self):          productName = self.autocompleteEntry.get()          productQuantity = self.quantityEntry.get()            #check if product name and quantity are not empty          if productName and productQuantity.isdigit():              quantity = int(productQuantity)              self.products.append([productName, quantity])                #update the display              self.updateProductList()                #clear entry fields after adding              self.autocompleteEntry.delete(0, customtkinter.END)              self.quantityEntry.delete(0, customtkinter.END)          else:              messagebox.showwarning("Input Error", "Please enter a valid product name and quantity")      #function to update the product list      def updateProductList(self):          #create a label and entry widget for each product in the list          self.clearProductList()          for i, product in enumerate(self.products):              if i==0:                  self.clearProductList()              countLabel = customtkinter.CTkLabel(self.productFrame, text=str(i+1))              countLabel.grid(row=i+2, column=0, padx=20, sticky="w", pady=10)              #name label with fixed width              nameLabel = customtkinter.CTkLabel(self.productFrame, text=product[0])              nameLabel.grid(row=i+2, column=1, padx=20, sticky="w", pady=10)              #quantity entry with fixed width              quantityEntryWidget = customtkinter.CTkEntry(self.productFrame)              quantityEntryWidget.grid(row=i+2, column=2, padx=20, sticky="w", pady=10)              quantityEntryWidget.insert(0, str(product[1]))  #insert the current quantity              #delete button to remove the product              print(self.products, i)              print(self.products[i])              deleteButton = customtkinter.CTkButton(self.productFrame, text="Delete", command=lambda i=i: self.deleteProductInDelivery(i))              deleteButton.grid(row=i+2, column=3, padx=20, sticky="w", pady=10)      #function to delete a product      def deleteProductInDelivery(self, index):          #remove product from the list          del self.products[index]          self.updateProductList()      #Function to clear product list      def clearProductList(self):          #clear the existing list          for widget in self.productFrame.winfo\_children():              if widget not in [self.productNumLabel, self.itemLabel, self.itemQuantityLabel, self.toolLabel]:                  widget.destroy()          self.deliveryDate.date\_entry.delete(0, customtkinter.END)          self.deliveryDate.date\_entry.insert(0, f"{datetime.now().day}/{self.month}/{datetime.now().year}")      #Function to confirm the delivery      def confirmDelivery(self):          if messagebox.askquestion(title='Confirm delivery', message="Do you wish to confirm the delivery?"):              try:                  #update stock levels and any other data here                  for product in self.products:                      #update stock level                      productID = self.productDB.getProductID(product[0])                      self.stockLevelDB.updateStockLevel(product[1], productID, True)                      #update last delivery date for product                      self.stockLevelDB.updateLastDelivery(json.dumps(self.deliveryDate.get\_date()), productID) #check json stuff                  #clear widgets once the delivery has been confirmed                  self.uiWidgetClearer()                  self.clearProductList()                  self.products = []              except Exception as error:                  print(f"error encountered on delivery confirmation: {error}")                  return False          else:              pass | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – stockCountingUI method | Requirement(s) met: **M1, M1A, M1B** | Technicality: Group B |
| def stockCountingUI(self, tab\_='Stock counting'):          self.tab\_ = tab\_          #create the autocomplete search for a product          self.findStockCountProductLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Search product:")          self.findStockCountProductLabel.grid(row=0, column=0, padx=(20), pady=20, sticky='w')          self.stockCountAutocompleteEntry = AutocompleteEntry(self.tabview.tab(tab\_), width=500, placeholder\_text='Search product...')            self.stockCountAutocompleteEntry.setSuggestions(self.productDB.getProductNames()) #set suggestions needs to be based on a call to the product table in the database          self.stockCountAutocompleteEntry.grid(row=0, column=1, padx=20, pady=20, columnspan=3, sticky='w')            self.stockCountQuantityLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Quantity: ")          self.stockCountQuantityLabel.grid(row=1, column=0, padx=(20, 20), pady=10, sticky='w')          self.stockCountQuantityEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="x")          self.stockCountQuantityEntry.grid(row=1, column=1, padx=(20, 20), pady=10, sticky='w')          self.addProduct = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Add product", command=self.addStockCountProductToDelivery)          self.addProduct.grid(row=1, column=2, padx=20, pady=10)          #create a seperator to distuinguish between sections          stockSeperator = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")          stockSeperator.grid(row=2, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')          #scrollable frame for added products          self.stockCountProducts = []          self.stockCountProductFrame = customtkinter.CTkScrollableFrame(master=self.tabview.tab(tab\_), width=300, height=200, corner\_radius=0, fg\_color="transparent")          self.stockCountProductFrame.grid(row=3, column=0, sticky="nsew", columnspan=6)          self.stockCountProductNumLabel = customtkinter.CTkLabel(self.stockCountProductFrame, text="Item num", fg\_color="transparent")          self.stockCountProductNumLabel.grid(row=0, column=0, padx=(20), pady=20, sticky='w')          self.stockCountitemLabel = customtkinter.CTkLabel(self.stockCountProductFrame, text="Date", fg\_color="transparent")          self.stockCountitemLabel.grid(row=0, column=1, padx=(20), pady=20, sticky='w')          self.stockCountitemQuantityLabel = customtkinter.CTkLabel(self.stockCountProductFrame, text="Quantity", fg\_color="transparent")          self.stockCountitemQuantityLabel.grid(row=0, column=2, padx=(20), pady=20, sticky='w')          self.stockCounttoolLabel = customtkinter.CTkLabel(self.stockCountProductFrame, text="Tool")          self.stockCounttoolLabel.grid(row=0, column=3, padx=(20), pady=20, sticky='w')            #create a seperator to distuinguish between sections          stockCountseperator = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")          stockCountseperator.grid(row=7, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')          self.confirmStockCountButton = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Confirm stock count", command=self.confirmStockCount)          self.confirmStockCountButton.grid(row=8, column=0, padx=20, pady=10) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – stockCountingUI functionality | Requirement(s) met: **M1, M1A, M1B** | Technicality: Group B |
| def addStockCountProductToDelivery(self):          productName = self.stockCountAutocompleteEntry.get()          productQuantity = self.stockCountQuantityEntry.get()            #check if product name and quantity are not empty          if productName and productQuantity.isdigit():              quantity = int(productQuantity)              self.stockCountProducts.append([productName, quantity])                #update the display              self.updateStockCountList()                #clear entry fields after adding              self.stockCountAutocompleteEntry.delete(0, customtkinter.END)              self.stockCountQuantityEntry.delete(0, customtkinter.END)          else:              messagebox.showwarning("Input Error", "Please enter a valid product name and quantity")      #function to update the product list      def updateStockCountList(self):          #create a label and entry widget for each product in the list          self.clearStockCountList()          for i, stockCountProduct in enumerate(self.stockCountProducts):              if i==0:                  self.clearStockCountList()              stockCountLabel = customtkinter.CTkLabel(self.stockCountProductFrame, text=str(i+1))              stockCountLabel.grid(row=i+2, column=0, padx=20, sticky="w", pady=10)              #name label with fixed width              nameLabel = customtkinter.CTkLabel(self.stockCountProductFrame, text=stockCountProduct[0])              nameLabel.grid(row=i+2, column=1, padx=20, sticky="w", pady=10)              #quantity entry with fixed width              quantityEntryWidget = customtkinter.CTkEntry(self.stockCountProductFrame)              quantityEntryWidget.grid(row=i+2, column=2, padx=20, sticky="w", pady=10)              quantityEntryWidget.insert(0, str(stockCountProduct[1]))  #insert the current quantity              #delete button to remove the product              print(self.stockCountProducts, i)              print(self.stockCountProducts[i])              deleteButton = customtkinter.CTkButton(self.stockCountProductFrame, text="Delete", command=lambda i=i: self.deleteProductInStockCountList(i))              deleteButton.grid(row=i+2, column=3, padx=20, sticky="w", pady=10)      #Function to delete a product      def deleteProductInStockCountList(self, index):          #Remove product from the list          del self.stockCountProducts[index]          self.updateStockCountList()      #function to clear product list      def clearStockCountList(self):          #clear the existing list          for widget in self.stockCountProductFrame.winfo\_children():              if widget not in [self.stockCountProductNumLabel, self.stockCountitemLabel, self.stockCountitemQuantityLabel, self.stockCounttoolLabel]:                  widget.destroy()      def confirmStockCount(self):          if messagebox.askquestion(title='Confirm stockcount', message="Do you wish to confirm the stockcount?"):              try:                  #update stock levels and any other data here                  for stockCountProduct in self.stockCountProducts:                      #update stock level                      productID = self.productDB.getProductID(stockCountProduct[0])                      self.stockLevelDB.updateStockLevel(stockCountProduct[1], productID)                  #clear widgets once the stockcount has been confirmed                  self.uiWidgetClearer()                  self.clearStockCountList()              except Exception as error:                  print(f"error encountered on stock count confirmation: {error}")                  return False          else:              pass | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – dataViewUI method | Requirement(s) met: **M1C, M1D, M5A** | Technicality: Group B |
| def dataViewUI(self, tab\_='Data view'):  self.tab\_ = tab\_  self.dataViewTabs = self.DBHandler.getTables()  self.dataViewTabs.pop(self.dataViewTabs.index('users')) #remove the user table from data that can be displayed  print(self.dataViewTabs)  try: #windows based db table names are lowercase  self.dataViewTabs.pop(self.dataViewTabs.index('stocklevelhistory')) #remove the stockLevelHistory table from data that can be displayed  self.dataViewTabs.pop(self.dataViewTabs.index('weeklyreportrecords')) #^^^^  except: #mac data view tab names are case sensitive for some reason  self.dataViewTabs.pop(self.dataViewTabs.index('stockLevelHistory')) #remove the stockLevelHistory table from data that can be displayed  self.dataViewTabs.pop(self.dataViewTabs.index('weeklyReportRecords')) #^^^^  self.dataViewTabView = customtkinter.CTkTabview(self.tabview.tab(tab\_))  self.dataViewTabView.grid(row=1, column=0, pady=(50,50), padx=(50, 50))  self.searchEntries = []  self.dataSets = []  self.displayTables = []  #each search entry for each tab will produce suggestions based on the data in these respective columns. The index is the number of the column to be used when the the search button command is called  searchEntrySuggestionsColumns = [  ["product\_name", 2],  ["stock\_id", 0],  ["supplier\_name", 1],  ["waste\_id", 0],  ]  #add table tabs to tabview  for i, \_tab in enumerate(self.dataViewTabs):  self.dataViewTabView.add(\_tab)  self.seeTableData(\_tab, searchEntrySuggestionsColumns[i][0], searchEntrySuggestionsColumns[i][1], i) #set the individual dataview UI for each respective table and its UI | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – dataViewUI functionality | Requirement(s) met: **M1C, M1D, M6B** | Technicality: Group A |
| #function to display the data inside the current table  def seeTableData(self, tab\_\_, searchEntrySuggestions, columnIndex, counter):  self.tabbbb = tab\_\_  self.counter = counter  #search entry to search the tableValues list  self.searchEntry = AutocompleteEntry(self.dataViewTabView.tab(tab\_\_), placeholder\_text="search...", width=400)  #sanitise columnData that was originally in tuple form and convert it into a string form  suggestions = self.DBHandler.getColumnData(searchEntrySuggestions, tab\_\_)  for i, suggestion in enumerate(suggestions):  suggestions[i] = str(reduce(lambda x, y: str(x) + ' ' + str(y), suggestion))    #set the suggestions for the auto compplete based on the sanitsed data  self.searchEntry.setSuggestions(suggestions)  self.searchEntry.grid(row=0, column=0, padx=20, pady=30, sticky='nsew')  #add search entry to search entries 2d list alongisde counter  self.searchEntries.append([self.searchEntry, counter])  self.searchButton = customtkinter.CTkButton(self.dataViewTabView.tab(tab\_\_), text="Search 🔍", command=lambda:self.searchButtonAlgo(self.searchEntries[counter][0].get(), columnIndex, self.dataSets[counter], self.displayTables[counter], tab\_\_))  self.searchButton.grid(row=0, column=1, sticky='nsew', padx=20, pady=30)  self.xy\_frame = CTkXYFrame(self.dataViewTabView.tab(tab\_\_), width=600, height=150)  self.xy\_frame.grid(row=2, column=0, sticky="nsew", columnspan=6)  #display the table of values  self.displayTable = CTkTable(self.xy\_frame, values=self.getTableData(self.tabbbb), header\_color="#1F538D")  self.displayTable.grid(row=0, column=0)  self.displayTables.append(self.displayTable)  def getTableData(self, currentTab):  #this needs to contain the database in a 2d list  self.tableValues = [self.DBHandler.getColumnNames(currentTab)]  self.dataSets.append(self.tableValues)  for row in self.DBHandler.getData(currentTab):  listVersion = list(row)  for i, listItem in enumerate(listVersion):  if type(listItem) is bytearray: #if the listItem is a byteArray (aka supplier dates as its stored in json), remove the byteArray prefixes  listVersion[i] = str(listItem.decode("utf-8"))  self.tableValues.append(listVersion)  return self.tableValues    def searchButtonAlgo(self, itemToFind, column, dataSet, table, tab):  self.graphVisualiser = CheckStockCount()  for i, row in enumerate(dataSet):  if str(row[int(column)]) == str(itemToFind):  print(row)  table.select\_row(row=i)    if tab == "products":  self.visualizeButtonLabel = customtkinter.CTkLabel(self.dataViewTabView.tab(tab), text="Stock level trends:")  self.visualizeButtonLabel.grid(row=3, column=0, padx=(10, 20), pady=20, sticky='w')  self.visualizeButton = customtkinter.CTkButton(self.dataViewTabView.tab(tab), text="Visualize ", command=lambda:self.visualize(itemToFind))  self.visualizeButton.grid(row=3, column=1, pady=20, padx=(10,20), sticky="w")  if tab == "waste":  if row[4] == 0: #if waste isnt resolved, then allow user to resolve by updating waste\_dealt\_with value to 1  self.changeResolvementStatusLabel = customtkinter.CTkLabel(self.dataViewTabView.tab(tab), text="Update resolvement status:")  self.changeResolvementStatusLabel.grid(row=4, column=0, padx=(10), pady=20, sticky='w')  self.changeResolvementStatusButton = customtkinter.CTkButton(self.dataViewTabView.tab(tab), text="Change", command=lambda:self.wasteDB.updateWasteResolvementValue(row[0]))  self.changeResolvementStatusButton.grid(row=4, column=1, pady=20, padx=(10,20), sticky="w")    else:  self.resolvementStatusLabel = customtkinter.CTkLabel(self.dataViewTabView.tab(tab), text="Waste already resolved")  self.resolvementStatusLabel.grid(row=4, column=0, padx=10, pady=20, sticky='w')  else:  table.deselect\_row(row=i)  def visualize(self, itemToFind):  xAxisVals = []  xAxisValsPrettified = []  yAxisVals = []  for row in self.stockLevelHistoryDB.getGraphValues(itemToFind):  xAxisVals.append(row[0])  xAxisValsPrettified.append(str(row[0])[:-9])  yAxisVals.append(row[1])  x = xAxisVals  y = yAxisVals  #create a line chart  plt.figure(figsize=(8, 6))  plt.plot(x, y, marker='o', linestyle='-')  #add annotations  for i, (xi, yi) in enumerate(zip(x, y)):  plt.annotate(f'({yi})', (xi, yi), textcoords="offset points", xytext=(0, 10), ha='center')  #add title and labels  plt.title(f"{itemToFind} stock level")  plt.xlabel("Date")  plt.ylabel("Stock level")  #display grid  plt.grid(True)  #show the plot  plt.show() | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – addProductUI method | Requirement(s) met: **M2, M2A, M2B** | Technicality: Group B |
| def addProductUI(self, tab\_='Add product'):  #you need to create a product database and then select all products in order to be able to give values for the value list below  self.tab\_ = tab\_  self.chooseSupplierLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Choose supplier:", anchor="w")  self.chooseSupplierLabel.grid(row=0, column=0, padx=(20, 20), pady=20, sticky='w')    #if no data in table, CTKOptionMenu throws an error, so try except block creates failure lable if this issue is encountered  try:  self.chooseSupplier2 = customtkinter.CTkOptionMenu(self.tabview.tab(tab\_), dynamic\_resizing=False, values=self.supplierDB.getSupplierNames(), width=200) #values list should be taken from a database call once the supplier database is created  self.chooseSupplier2.grid(row=0, column=1, padx=20, pady=20)    except Exception as error:  print(error)  self.noSupplierLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="No suppliers found", anchor="w")  self.noSupplierLabel.grid(row=0, column=1, padx=(20, 20), pady=20, sticky='w')  #create entry widgets and their respective labels for data input  self.productNameLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Product name: ")  self.productNameLabel.grid(row=0, column=2, padx=(20, 20), pady=20, sticky='w')  self.productNameEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="product name...")  self.productNameEntry.grid(row=0, column=3, padx=(20, 20), pady=20, sticky='w')  self.productBuyPriceEntryLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Product buy price: ")  self.productBuyPriceEntryLabel.grid(row=1, column=0, padx=(20, 20), pady=20, sticky='w')  self.productBuyPriceEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="buy price...")  self.productBuyPriceEntry.grid(row=1, column=1, padx=(20, 20), pady=20, sticky='w')  self.productSellPriceEntryLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Product sell price: ")  self.productSellPriceEntryLabel.grid(row=1, column=2, padx=(20, 20), pady=20, sticky='w')  self.productSellPriceEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="sell price...")  self.productSellPriceEntry.grid(row=1, column=3, padx=(20, 20), pady=20, sticky='w')  self.productPSLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Product pack size: ")  self.productPSLabel.grid(row=2, column=0, padx=(20, 20), pady=20, sticky='w')  self.productPSEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="pack size...")  self.productPSEntry.grid(row=2, column=1, padx=(20, 20), pady=20, sticky='w')  self.productWeightLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Product weight: ")  self.productWeightLabel.grid(row=2, column=2, padx=(20, 20), pady=20, sticky='w')  self.productWeightEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="weight...")  self.productWeightEntry.grid(row=2, column=3, padx=(20, 20), pady=20, sticky='w')  self.minimumStockLevelLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Min stock level: ")  self.minimumStockLevelLabel.grid(row=3, column=0, padx=(20, 20), pady=20, sticky='w')  self.minimumStockLevelEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="min stock level...")  self.minimumStockLevelEntry.grid(row=3, column=1, padx=(20, 20), pady=20, sticky='w')  self.reorderStockLevelLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Re-order level: ")  self.reorderStockLevelLabel.grid(row=3, column=2, padx=(20, 20), pady=20, sticky='w')  self.reorderStockLevelEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="re-order level...")  self.reorderStockLevelEntry.grid(row=3, column=3, padx=(20, 20), pady=20, sticky='w')  self.productDescriptionLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Product description: ")  self.productDescriptionLabel.grid(row=4, column=0, padx=(20, 20), pady=20, sticky='w')  self.productDescriptionEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), width=518, placeholder\_text="product description...")  self.productDescriptionEntry.grid(row=4, column=1, padx=(20, 20), pady=20, sticky='w', columnspan=5)  self.confirmAddProduct = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Confirm add product", command=self.confirmAddproductProcess)  self.confirmAddProduct.grid(row=5, column=0, padx=20, pady=20) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – addProductUI functionality | Requirement(s) met: **M2, M2A, M2B** | Technicality: Group B |
| def confirmAddproductProcess(self):  try:  if messagebox.askquestion(title='Confirm add product', message="Do you wish to confirm this new product?"):  #create the new product - supplier\_id needs to be found first#  supplierID = self.supplierDB.getSupplierID(self.chooseSupplier2.get())  self.productDB.createProduct(supplierID[0], self.productNameEntry.get(), self.productDescriptionEntry.get(), self.productPSEntry.get(), self.productWeightEntry.get(), self.productBuyPriceEntry.get(), self.productSellPriceEntry.get())  stockLevelProductID = self.productDB.getProductID(self.productNameEntry.get())  self.stockLevelDB.addStockLevelData(stockLevelProductID, self.minimumStockLevelEntry.get(), self.reorderStockLevelEntry.get())  self.autocompleteEntry.setSuggestions(self.productDB.getProductNames())  for widget in [self.productNameEntry, self.productBuyPriceEntry, self.productSellPriceEntry, self.productPSEntry, self.productWeightEntry, self.minimumStockLevelEntry, self.reorderStockLevelEntry, self.productDescriptionEntry]:  try:  widget.delete(0, customtkinter.END)  widget.\_activate\_placeholder()  widget.focus()    #this handles the event that an entry widget doesnt register the placeholder text, such as an auto\_complete entry  except Exception as error:  print(f"e: {error}")  continue  #resume with app if "no" option is selected  else:  pass    except Exception as error:  print(error)  messagebox.showerror("Error", f"An error occurred! Please try again. If this issue persits, please contact the maintainer. Error {error}")  #limits entry widget to 200 characters by default  def limit\_entry(self, limit=200, \*args):  current\_text = self.limiter.get()    if len(current\_text) > limit:  self.limiter.set(current\_text[:limit]) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – addSupplierUI method | Requirement(s) met: **M3, M3A** | Technicality: Group B |
| def addSupplierUI(self, tab\_='Add supplier'):  self.tab\_ = tab\_  self.chooseSupplierLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Supplier name:", anchor="w")  self.chooseSupplierLabel.grid(row=0, column=0, padx=(20, 20), pady=20, sticky='w')  self.suppliertNameEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="supplier name...")  self.suppliertNameEntry.grid(row=0, column=1, padx=(20, 20), pady=20, sticky='w')  self.supplierDescriptionLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Supplier description: ")  self.supplierDescriptionLabel.grid(row=2, column=0, padx=(20, 20), pady=20, sticky='w')  self.supplierDescriptionEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), width=500, placeholder\_text="supplier description...")  self.supplierDescriptionEntry.grid(row=2, column=1, padx=(20, 20), pady=20, sticky='w', columnspan=5)  #you need supplier dates here, consider storing this as a list in a JSON format  self.supplierDates = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Supplier date: ")  self.supplierDates.grid(row=3, column=0, padx=(20, 20), pady=10, sticky='w')  self.supplierDatesEntry = CTkDatePicker(self.tabview.tab(tab\_))  self.supplierDatesEntry.grid(row=3, column=1, padx=(15, 20), pady=10, sticky='w')  self.addSupplierDate = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Add supplier delivery date", command=self.addSupplierDeliveryDate)  self.addSupplierDate.grid(row=3, column=2, padx=20, pady=10)  #create a seperator to distuinguish between sections  seperator = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")  seperator.grid(row=4, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')  #scrollable frame for added supplier dates  self.supplierDates = []  self.supplierDateFrame = customtkinter.CTkScrollableFrame(master=self.tabview.tab(tab\_), width=300, height=200, corner\_radius=0, fg\_color="transparent")  self.supplierDateFrame.grid(row=5, column=0, sticky="nsew", columnspan=6)  self.supplierDateNumLabel = customtkinter.CTkLabel(self.supplierDateFrame, text="Date num", fg\_color="transparent")  self.supplierDateNumLabel.grid(row=0, column=0, padx=(20), pady=20, sticky='w')  self.dateLabel = customtkinter.CTkLabel(self.supplierDateFrame, text="Item", fg\_color="transparent")  self.dateLabel.grid(row=0, column=1, padx=(20), pady=20, sticky='w')  self.toolLabel = customtkinter.CTkLabel(self.supplierDateFrame, text="Tool")  self.toolLabel.grid(row=0, column=3, padx=(20), pady=20, sticky='w')    #create a seperator to distuinguish between sections  seperator2 = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")  seperator2.grid(row=6, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')  self.confirmAddSupplier = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Confirm add supplier", command=self.confirmAddSupplierProcess)  self.confirmAddSupplier.grid(row=7, column=0, padx=20, pady=20) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – addSuppplierUI functionality | Requirement(s) met: **M3, M3A, M3B** | Technicality: Group B |
| #Creates the new supplier and clears all entry widgets  def confirmAddSupplierProcess(self):  try:  if messagebox.askquestion(title='Confirm add supplier', message="Do you wish to confirm this new supplier?"):  #create the new supplier  self.supplierDB.createSupplier(self.suppliertNameEntry.get(), self.supplierDescriptionEntry.get(), json.dumps(self.supplierDates))    #reset widgets  self.uiWidgetClearer()    #update supplier option menus  self.chooseSupplier1.configure(values=self.supplierDB.getSupplierNames())  self.chooseSupplier2.configure(values=self.supplierDB.getSupplierNames())  #delete supplier delivery date fields upon successful supplier creation  self.supplierDates = []  self.updateSupplierDeliveryDateList()  #resume with app if "no" option is selected  else:  pass  except Exception as error:  print(error)  messagebox.showerror("Error", f"An error occurred! Please try again. If this issue persits, please contact the maintainer. Error {error}")  def addSupplierDeliveryDate(self):  deliveryDate = self.supplierDatesEntry.get\_date()    #check supplier date are not empty  if deliveryDate:  self.supplierDates.append(deliveryDate)  self.updateSupplierDeliveryDateList()  self.supplierDatesEntry.date\_entry.delete(0, customtkinter.END)    else:  messagebox.showwarning("Input Error", "Please enter a valid delivery date")  def updateSupplierDeliveryDateList(self):  #create widgets for each supplier date  self.clearSupplierDeliveryDateList()  for i, supplierDate in enumerate(self.supplierDates):  if i==0:  self.clearSupplierDeliveryDateList()  count\_label = customtkinter.CTkLabel(self.supplierDateFrame, text=str(i+1))  count\_label.grid(row=i+2, column=0, padx=20, sticky="w", pady=10)  #name label with fixed width  name\_label = customtkinter.CTkLabel(self.supplierDateFrame, text=supplierDate)  name\_label.grid(row=i+2, column=1, padx=20, sticky="w", pady=10)  #delete button to remove the supplier date  print(self.supplierDates, i)  print(self.supplierDates[i])  delete\_button = customtkinter.CTkButton(self.supplierDateFrame, text="Delete", command=lambda i=i: self.deleteSupplierDate(i))  delete\_button.grid(row=i+2, column=3, padx=20, sticky="w", pady=10)  def clearSupplierDeliveryDateList(self):  #clear the existing list  for widget in self.supplierDateFrame.winfo\_children():  if widget not in [self.supplierDateNumLabel, self.dateLabel, self.toolLabel]:  widget.destroy()  def deleteSupplierDate(self, index):  #remove supplier date from the list  del self.supplierDates[index]  self.updateSupplierDeliveryDateList() | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – wasteUI method | Requirement(s) met: **M6, M6A, M6B** | Technicality: Group B |
| def wasteUI(self, tab\_='Waste'):  self.tab\_ = tab\_  self.findWasteProductLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Search product:")  self.findWasteProductLabel.grid(row=0, column=0, padx=(20), pady=20, sticky='w')  self.findWasteProductEntry = AutocompleteEntry(self.tabview.tab(tab\_), width=500, placeholder\_text='search product...')    self.findWasteProductEntry.setSuggestions(self.productDB.getProductNames()) #set suggestions needs to be based on a call to the product table in the database  self.findWasteProductEntry.grid(row=0, column=1, padx=20, pady=20, columnspan=3, sticky='w')    self.wasteDescriptionLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Waste description: ")  self.wasteDescriptionLabel.grid(row=1, column=0, padx=(20, 20), pady=20, sticky='w')  self.wasteDescriptionEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), width=500, placeholder\_text="waste description...")  self.wasteDescriptionEntry.grid(row=1, column=1, padx=(20, 20), sticky='w', columnspan=5)  self.wasteQuantityLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Quantity: ")  self.wasteQuantityLabel.grid(row=2, column=0, padx=(20, 20), pady=10, sticky='w')  self.wasteQuanitityEntry = customtkinter.CTkEntry(self.tabview.tab(tab\_), placeholder\_text="x")  self.wasteQuanitityEntry.grid(row=2, column=1, padx=(20, 20), pady=10, sticky='w')  self.wasteStateCheckboxVar = customtkinter.StringVar(value=False)  self.wasteStateCheckbox = customtkinter.CTkCheckBox(self.tabview.tab(tab\_), text="Dealt with",variable=self.wasteStateCheckboxVar, onvalue=True, offvalue=False)  self.wasteStateCheckbox.grid(row=2, column=2)  self.addWasteProduct = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Add waste product", command=self.addWasteProductToList)  self.addWasteProduct.grid(row=2, column=3, padx=20, pady=10)  #create a seperator to distuinguish between sections  wasteSeperator1 = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")  wasteSeperator1.grid(row=3, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')  #scrollable frame for added products  self.wasteProducts = []  self.wasteProductFrame = customtkinter.CTkScrollableFrame(master=self.tabview.tab(tab\_), width=300, height=200, corner\_radius=0, fg\_color="transparent")  self.wasteProductFrame.grid(row=4, column=0, sticky="nsew", columnspan=6)  self.wasteProductNumLabel = customtkinter.CTkLabel(self.wasteProductFrame, text="Item num", fg\_color="transparent")  self.wasteProductNumLabel.grid(row=0, column=0, padx=(20), pady=20, sticky='w')  self.wasteItemLabel = customtkinter.CTkLabel(self.wasteProductFrame, text="Item", fg\_color="transparent")  self.wasteItemLabel.grid(row=0, column=1, padx=(20), pady=20, sticky='w')  self.wasteItemQuantityLabel = customtkinter.CTkLabel(self.wasteProductFrame, text="Quantity", fg\_color="transparent")  self.wasteItemQuantityLabel.grid(row=0, column=2, padx=(20), pady=20, sticky='w')  self.wasteStatusLabel = customtkinter.CTkLabel(self.wasteProductFrame, text="Status", fg\_color="transparent")  self.wasteStatusLabel.grid(row=0, column=3, padx=(20), pady=20, sticky='w')  self.wasteToolLabel = customtkinter.CTkLabel(self.wasteProductFrame, text="Tool")  self.wasteToolLabel.grid(row=0, column=4, padx=(20), pady=20, sticky='w')    #create a seperator to distuinguish between sections  wasteSeperator2 = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray")  wasteSeperator2.grid(row=7, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')  self.confirmWasteButton = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Confirm waste products", command=self.confirmAddWasteProductProcess)  self.confirmWasteButton.grid(row=8, column=0, padx=20, pady=10) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – wasteUI functionality | Requirement(s) met: **M6, M6A, M6B** | Technicality: Group B |
| def confirmAddWasteProductProcess(self):  try:  if messagebox.askquestion(title='Confirm add waste product(s)', message="Do you wish to confirm this waste?"):  #db call to create waste products  for wasteProduct in self.wasteProducts:  product\_id = wasteProduct[0]  supplier\_id = self.productDB.getRespectiveSupplerID(product\_id)  wasteReason = wasteProduct[1]  wasteQuantity = int(wasteProduct[2])  wasteState = wasteProduct[3]  #create waste product in the database  self.wasteDB.createWasteProduct(self.productDB.getProductID(product\_id), supplier\_id, wasteReason, wasteState)  #update stock level (subtracting waste quantity)  self.stockLevelDB.updateStockLevel(wasteQuantity\*-1, self.productDB.getProductID(product\_id))  #reset widgets  self.uiWidgetClearer()    self.wasteProducts = []  self.updateWasteProductList(True)  #resume with app if "no" option is selected  else:  pass  except Exception as error:  print(error)  messagebox.showerror("Error", f"An error occurred! Please try again. If this issue persits, please contact the maintainer. Error {error}")  def addWasteProductToList(self):  wasteProduct = [self.findWasteProductEntry.get(), self.wasteDescriptionEntry.get(), self.wasteQuanitityEntry.get(), self.wasteStateCheckboxVar.get()]    #make sure we have all the data to create a waste product  tally = 0  for entryWidgetData in wasteProduct:  if not(entryWidgetData == '') and wasteProduct[2].isdigit(): #make sure the quantity entry is a number  tally += 1  if tally == 4:  self.wasteProducts.append(wasteProduct)  self.updateWasteProductList()  self.uiWidgetClearer()    #reset entry widgets for a cleaner look and to not retain previously entered product data as this is already stored in the waste product list  self.findWasteProductEntry.delete(0, customtkinter.END)  self.wasteDescriptionEntry.delete(0, customtkinter.END)  self.wasteQuanitityEntry.delete(0, customtkinter.END)  self.wasteStateCheckbox.deselect() #make sure the checkbox is in the off state    else:  messagebox.showwarning("Input Error", "Please enter a valid waste product")  def updateWasteProductList(self, clear\_=False):  if clear\_:  self.clearWasteProductList()  for i, wasteProduct in enumerate(self.wasteProducts):  if i==0:  self.clearWasteProductList()  count\_label = customtkinter.CTkLabel(self.wasteProductFrame, text=str(i+1))  count\_label.grid(row=i+2, column=0, padx=20, sticky="w", pady=10)  #Name label with fixed width  name\_label = customtkinter.CTkLabel(self.wasteProductFrame, text=wasteProduct[0])  name\_label.grid(row=i+2, column=1, padx=20, sticky="w", pady=10)  #Delete button to remove the supplier date  print(self.wasteProducts, i)  print(self.wasteProducts[i])  quantity\_label = customtkinter.CTkLabel(self.wasteProductFrame, text=self.wasteProducts[i][2])  quantity\_label.grid(row=i+2, column=2, padx=20, sticky="w", pady=10)  status\_label = customtkinter.CTkLabel(self.wasteProductFrame, text=self.wasteProducts[i][3])  status\_label.grid(row=i+2, column=3, padx=20, sticky="w", pady=10)  delete\_button = customtkinter.CTkButton(self.wasteProductFrame, text="Delete", command=lambda i=i: self.deleteWasteProduct(i))  delete\_button.grid(row=i+2, column=4, padx=20, sticky="w", pady=10)  def clearWasteProductList(self):  #clear the existing list  for widget in self.wasteProductFrame.winfo\_children():  if widget not in [self.wasteProductNumLabel, self.wasteItemLabel, self.wasteItemQuantityLabel, self.wasteStatusLabel]:  widget.destroy()  def deleteWasteProduct(self, index):  # Remove waste product from the list  del self.wasteProducts[index]  self.updateWasteProductList() | | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – weeklyReportUI method | Requirement(s) met: **M9, M9F** | Technicality: Group B |
| def weeklyReportUI(self, tab\_='Weekly report'):  self.tab\_ = tab\_  #this should be a list of potential weeks  reportsSortedByWeek = []  prevWeeklyReportsIDS, self.weeklyReportsData = self.weeklyReportDB.getWeeklyReportsAsList()  for report in prevWeeklyReportsIDS:  reportsSortedByWeek.append(f"Weekly report - {report[0]}")  reportsSortedByWeek = reportsSortedByWeek[::-1] #flip the order of weekly report to most recent first to advoid user having to scroll too far  #this should be a list of potential weeks  self.seePreviousWeeklyReportButtonCombobox = customtkinter.CTkOptionMenu(self.tabview.tab(tab\_), values=reportsSortedByWeek)  self.seePreviousWeeklyReportButtonCombobox.grid(row=0, column=0, padx=(20, 20), pady=20, sticky='w')  self.seePreviousWeeklyReportButton = customtkinter.CTkButton(self.tabview.tab(tab\_), text="See previous report", command=lambda:self.seePrevReport(self.seePreviousWeeklyReportButtonCombobox.get()))  self.seePreviousWeeklyReportButton.grid(row=0, column=1, padx=(20, 20), pady=20, sticky='w')  self.weeklyReportFrame = customtkinter.CTkScrollableFrame(master=self.tabview.tab(self.tab\_), width=290, height=100, corner\_radius=0, fg\_color="transparent")  self.weeklyReportFrame.grid(row=1, column=0, sticky="nsew", columnspan=6)  seperator = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray", width=660)  seperator.grid(row=2, column=0, columnspan=10, padx=20, pady=20)  self.sendEmailVar = customtkinter.StringVar(value=False)  self.sendEmailCheckbox = customtkinter.CTkCheckBox(self.tabview.tab(tab\_), text="Send email breakdown",variable=self.sendEmailVar, onvalue=True, offvalue=False)  self.sendEmailCheckbox.grid(row=3, column=0, padx=(20, 20), pady=20, sticky='w')  self.sendEmailCheckbox.select() #set default state as selected  self.produceTxtOutputVar = customtkinter.StringVar(value=False)  self.produceTxtOutput = customtkinter.CTkCheckBox(self.tabview.tab(tab\_), text="Produce .txt output",variable=self.sendEmailVar, onvalue=True, offvalue=False)  self.produceTxtOutput.grid(row=3, column=1, padx=(20, 20), pady=20, sticky='w')  self.produceTxtOutput.select() #set default state as selected  currentDate = datetime.today()  currentDateMinusWeek = (currentDate - timedelta(weeks=1)).strftime("%d/%m/%Y")  self.generateWeeklyReportButton = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Generate weekly report", command=lambda:self.generateWeeklyReport(currentDateMinusWeek, currentDate))  self.generateWeeklyReportButton.grid(row=4, column=0, padx=(20, 20), pady=20, sticky='w')  self.reportInfoLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text=f"\*local reports will generate at {self.\_appLocation}") #tell the user where the report is generated by getting the working directory  self.reportInfoLabel.grid(row=4, column=1, padx=(20, 20), pady=20, sticky='w')  seperator2 = customtkinter.CTkFrame(self.tabview.tab(tab\_), height=2, fg\_color="gray", width=660)  seperator2.grid(row=5, column=0, columnspan=10, padx=20, pady=20) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – weeklyReportUI functionality | Requirement(s) met: **M9, M9A, M9B, M9C, M9D, M9E** | Technicality: Group A |
| def seePrevReport(self, selected):  #delete any previous labels if there are any  for widget in self.weeklyReportFrame.winfo\_children():  widget.destroy()  date = selected[16:]  target = []  for report in self.weeklyReportsData:  if report[2].strftime("%d/%m/%Y") == date:  target.append(report)  for i, report in enumerate(target):  formattedReport = (f"Weekly report record ID: {report[0]}\n"  f"Product\_ID: {report[1]}\n"  f"Date of creation: {report[2].strftime('%Y-%m-%d %H:%M:%S')}\n"  f"Trend in stock level: {report[3]}\n"  f"Predicted stock levels for next week: {report[4].decode()}\n"  f"Revenue: £{report[5]:.2f}\n"  f"COGS: £{report[6]:.2f}\n"  f"Profit: £{report[7]:.2f}")    self.weeklyReportDataLabel = customtkinter.CTkLabel(self.weeklyReportFrame, text=f"Report record {i+1}) {formattedReport}") #tell the user where the report is generated by getting the working directory  self.weeklyReportDataLabel.grid(row=i, column=0, padx=(20, 20), pady=20, sticky='w')  def findDateRange(self, startDate, endDate):  if type(startDate) == str:  startDate = datetime.strptime(startDate, "%d/%m/%Y")  if type(endDate) == str:  endDate = datetime.strptime(endDate, "%d/%m/%Y")  dateRangeList = []  currentDate = startDate  while currentDate <= endDate:  dateRangeList.append(currentDate.strftime("%d/%m/%Y"))  currentDate += timedelta(days=1)  return dateRangeList  def generateWeeklyReport(self, startDate, endDate):  if messagebox.askquestion(title='Confirm generate weekly report', message="Do you wish to generate this weekly report?"):  #WEEKLY REPORT GENERATION==============================================================================================  dateRangeList = self.findDateRange(startDate, endDate)  productNames = self.productDB.getProductNames()  productDataMap = {} #dict to store product data  #fetch stock history data  self.DBHandler.cursor.execute("SELECT stock\_count, stock\_history\_product\_name, DATE\_FORMAT(date, '%d/%m/%Y'), action FROM stocklevelhistory")  allStockData = self.DBHandler.cursor.fetchall()  #put stock history into the dict  stockHistoryDict = {}  for stockCount, productName, dateStr, action in allStockData:  key = (productName, dateStr, action)  if key not in stockHistoryDict:  stockHistoryDict[key] = []  stockHistoryDict[key].append(stockCount)  #process data into productData2dList  for dateStr in dateRangeList:  for productName in productNames:  for action in ['delivery', 'count', 'waste']:  key = (productName, dateStr, action)  if key in stockHistoryDict:  stockCounts = stockHistoryDict[key]    if productName not in productDataMap:  productDataMap[productName] = [productName, [], [], []]    #extend the stock counts and corresponding actions  productDataMap[productName][1].extend(stockCounts)  productDataMap[productName][2].extend([dateStr] \* len(stockCounts))  productDataMap[productName][3].extend([action] \* len(stockCounts))  #conv dict vals into list  productData2dList = list(productDataMap.values())  for field in productData2dList:  print(field, end="\n")  weeklyReportData = []  for currentIndex, valueList in enumerate(productData2dList):  #add the productName to the current products data analysis  weeklyReportData.append([valueList[0]])  #create a dates and stockCounts list so that data can be analysed  dates = []  stockCounts = []  actions = []  for i in range(len(valueList[1])):  dateObj = datetime.strptime(valueList[2][i], "%d/%m/%Y")  dates.append(dateObj)  stockCounts.append(valueList[1][i])  actions.append(valueList[3][i])  #append date and subsequent stock count info to the weekly report for this product  newDates = []  for \_date in dates:  newDates.append(datetime.strftime(\_date, "%d/%m/%Y")) #reformat the dates    weeklyReportData[currentIndex].append(newDates)  weeklyReportData[currentIndex].append(stockCounts)  #convert dates to ints  startDate\_ = dates[0]  days = []  for \_date in dates:  days.append((\_date - startDate\_).days)    #calculate the trend in stock count (is it positive, negative, or stable?)  sumX = 0  sumY = 0  sumXY = 0  sumX2 = 0  for i in range(len(days)):  sumX += days[i]  sumY += stockCounts[i]  sumXY += days[i] \* stockCounts[i]  sumX2 += days[i] \*\* 2  #calc the gradient  numerator = (len(days) \* sumXY) - (sumX \* sumY)  denominator = (len(days) \* sumX2) - (sumX \*\* 2)  if denominator != 0:  gradient = numerator / denominator  else:  gradient = 0  if gradient > 0:  conclusion = "Increasing stock levels"    elif gradient < 0:  conclusion = "Decreasing stock levels"  else:  conclusion = "No significant change"    #add the type of trend in the stock level of the current product  weeklyReportData[currentIndex].append(conclusion)  #calculate the y intercept  if len(days) > 0:  intercept = (sumY - gradient \* sumX) / len(days)  else:  intercept = 0  #predict stock levels for the next 7 days  futureDays = []  futureStock = []  lastDay = days[-1] #last recorded day  for i in range(1, 7):  next\_day = lastDay + i  futureDays.append(next\_day)  futureStock.append(int(gradient \* next\_day + intercept))  #add the predicted stock level for the next week to the data analysis for the current product  weeklyReportData[currentIndex].append(futureStock)  #calculate profit margins for the current week for this product  self.DBHandler.cursor.execute("SELECT product\_buy\_price, product\_sell\_price FROM products WHERE product\_name = ?", (valueList[0],))  buyPrice, sellPrice = self.DBHandler.cursor.fetchone()  #calculate revenue, cost of good sale & net profit  totalRevenue = 0  totalCogs = 0  totalSold = 0  for i in range(len(stockCounts)):  if actions[i] == "count":  totalSold += stockCounts[i]  totalRevenue += stockCounts[i] \* sellPrice  totalCogs += stockCounts[i] \* buyPrice  elif actions[i] == "delivery":  continue  elif actions[i] == "waste":  totalSold += stockCounts[i]  totalRevenue -= stockCounts[i] \* sellPrice  totalCogs += stockCounts[i] \* buyPrice  #calc net profit  totalSold = totalRevenue - totalCogs  weeklyReportData[currentIndex].append({"revenue": totalRevenue, "cogs": totalCogs, "profit": totalSold})  #write record to weekly report table  for product in weeklyReportData:  product\_id = self.productDB.getProductID(product[0])  self.weeklyReportDB.addWeeklyReportRecord(product\_id, product[3], json.dumps(product[4]), product[5]['revenue'], product[5]['cogs'], product[5]['profit'])  #OPTIONAL STUFFS=======================================================================================================  #if either of the send email or produce .txt file checkboxes are selected, then generate the multiline string output and go from there  if self.produceTxtOutputVar.get() or self.sendEmailCheckbox.get():  totalInfo = """"""  for i, productReport in enumerate(weeklyReportData):  multiLineInfo = f""""""  if i != 0:  multiLineInfo += "<=====================================NEXT-PRODUCT========================================>\n"  multiLineInfo += f"PRODUCT-NAME: {productReport[0]}\n"    multiLineInfo += f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"  multiLineInfo += f"PRODUCT-STOCK-COUNTS:\n"  multiLineInfo += f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"    for j, date\_ in enumerate(productReport[1]):  multiLineInfo += f"{date\_}: {productReport[2][j]}\n"  multiLineInfo += f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"  multiLineInfo += f"LINEAR-REGRESSION-ANALYSIS: {productReport[3]}\n"  multiLineInfo += f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"  multiLineInfo += f"PREDICTED-STOCK-COUNT-FOR-NEXT-WEEK: \n-> {productReport[4]}\n"  multiLineInfo += f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"  multiLineInfo += f"<=========================================================================================>\n\n\n\n"  totalInfo += multiLineInfo  #local weekly reports are stored in a weekly reports folder, if it doesnt exist then make it  if self.produceTxtOutputVar.get():  reportPath = f"weeklyReports"  if not os.path.exists(reportPath):  os.makedirs(reportPath)  #create .txt file here  newWeeklyReportFilePath = f"weekly\_report\_{date.today().strftime('%d-%m-%Y')}.txt"  print(f"Generated file path: {newWeeklyReportFilePath}")  if not os.path.isfile(newWeeklyReportFilePath):  with open(newWeeklyReportFilePath, 'w') as file\_:  file\_.write(totalInfo)    #move weekly report file to weekly reports  shutil.move(newWeeklyReportFilePath, f"weeklyReports/{newWeeklyReportFilePath}")    else:  message = popUpWindow("Weekly report already exists")  message.create()  #user might want the report emailed, so do this here (however this is selected by default)  if self.sendEmailCheckbox.get():  #send email containing report here  emailAlert = appEmail()  emailAlert.sendEmail(self.\_defAlertEmail, f"Weekly report - {date.today().strftime('%d-%m-%Y')}", totalInfo) | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – settingsUI method & functionality | Requirement(s) met: **M12, M12A, M12B** | Technicality: Group B |
| def settingsUI(self, tab\_='Settings'):  self.tab\_ = tab\_  #user tools  self.userToolsLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="User tools:")  self.userToolsLabel.grid(row=0, column=0, padx=(20, 20), pady=20, sticky='w')  self.addUserButton = customtkinter.CTkButton(self.tabview.tab(tab\_), text="Create new user", command=self.addNewUser)  self.addUserButton.grid(row=1, column=0, sticky="nw", padx=(20, 20))  #configure settings button states  if int(self.userAccessLevel) != 1:  for settingsButton in [self.addUserButton]:  settingsButton.configure(state="disabled")  if int(self.userAccessLevel) == 1:  #show environment variables and edit them if the user is an admin  self.envVariableLabel = customtkinter.CTkLabel(self.tabview.tab(tab\_), text="Environment variables:")  self.envVariableLabel.grid(row=2, column=0, padx=(20, 20), pady=20, sticky='nw')    self.envVarLabels = {}  self.envVars = ["DB\_USERNAME", "DB\_PASSWORD", "DB\_HOST", "DB\_SCHEMA", "DEF\_EMAIL\_ADDR", "DEF\_EMAIL\_ADDR\_PASS", "DEF\_ALERT\_RECIPIENT\_EMAIL"]  rows = [3, 4, 5, 6, 7, 8, 9]  maxVarLen = max(len(var) for var in self.envVars)  dots\_ = 5  self.envVarEntries = {}  self.envVarButtons = {}  for i, variable in enumerate(self.envVars):  envValue = os.getenv(variable, "N/A")  dots = "." \* (maxVarLen + dots\_ - len(variable))  info = f"{i+1}) {variable} {dots} {envValue}"  #label to display current environment variable  self.envVarLabels[i] = customtkinter.CTkLabel(self.tabview.tab(tab\_), text=info, font=("Courier", 12))  self.envVarLabels[i].grid(row=rows[i], column=0, padx=(40, 10), pady=(10,10), sticky='w')  #entry field to input new value  self.envVarEntries[i] = customtkinter.CTkEntry(self.tabview.tab(tab\_), width=200, placeholder\_text="new value...")  self.envVarEntries[i].grid(row=rows[i], column=1, padx=(10, 10), sticky='w')  #tick button to update the variable  def updateEnvVar(var=variable, entryWidget=self.envVarEntries[i]):  newValue = entryWidget.get()  envVarPath="src/config/.env"  if newValue:  dotenv.set\_key(envVarPath, var, newValue, quote\_mode='always', export=False, encoding='utf-8')  print(f"Updated {var} -> {newValue}")  entryWidget.delete(0, customtkinter.END)  self.envVarButtons[i] = customtkinter.CTkButton(self.tabview.tab(tab\_), text="↻", width=30, command=updateEnvVar)  self.envVarButtons[i].grid(row=rows[i], column=2, padx=(10, 0), sticky='w')  def addNewUser(self):  user = newUser()  user.mainloop() | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – uiWidgetClearer method | Requirement(s) met: **S2, S2A** | Technicality: Group B |
| def uiWidgetClearer(self):  for widget in self.tabview.tab(self.tab\_).winfo\_children():  try:  if widget.cget('placeholder\_text'):  widget.delete(0, customtkinter.END)  widget.\_activate\_placeholder()  widget.focus()    except ValueError:  continue | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – isRowEmpty method | Requirement(s) met: **S2, S2A** | Technicality: Group B |
| #func to check if there are any widgets present in a row  def isRowEmpty(parent, rowNumber):  for widget in parent.winfo\_children():  info = widget.grid\_info()  if ‘row’ in info and info[‘row’] == rowNumber:  return False    return True | | |

|  |  |  |
| --- | --- | --- |
| mainApp.py – if \_\_name\_\_ == “\_\_main\_\_” | Requirement(s) met: **M7, M7A** | Technicality: Group A |
| if \_\_name\_\_ == “\_\_main\_\_”:  def runMainApp():  #initalise the databases logon database  initialiser = logonDBHandler()  initialiser.initializeDatabase()  #Run the UI  app = App(1)  app.mainloop()  def checkStockCounts():  runStockCheck = CheckStockCount()  runStockCheck.runStockLevelCheckAgainstMinimum()  mainAppThread = threading.Thread(target=runMainApp)  checkStockCountThread = threading.Thread(target=checkStockCounts)  #for running on mac, the UI needs to run on the main thread, otherwise there will be an exception  if threading.current\_thread() is threading.main\_thread():  runMainApp()  else:  threading.main\_thread().run(runMainApp)  checkStockCountThread.start()  checkStockCountThread.join() | | |

*config-folder source*

|  |  |  |
| --- | --- | --- |
| .env – environment variables | Requirement(s) met: **S6, S6a** | Technicality: Group C |
| DB\_USERNAME='root'  DB\_PASSWORD='BeltMadness3'  DB\_HOST='127.0.0.1'  DB\_SCHEMA='nea'  DEF\_EMAIL\_ADDR='stockmonitoringassistant@gmail.com'  DEF\_EMAIL\_ADDR\_PASS='ytav hxke tgqf ofel'  DEF\_ALERT\_RECIPIENT\_EMAIL='sbenf999@outlook.com' | | |

|  |  |  |
| --- | --- | --- |
| staticSetUpState.txt – program set up status | Requirement(s) met: n/a | Technicality: Group C |
| True | | |

*dbHandling-folder source*

|  |  |  |
| --- | --- | --- |
| dbHandler.py file – parent class | Requirement(s) met: **S3** | Technicality: Group A |
| *\*DISCLAIMER: THIS IS A MODIFIED VERSION OF THE ORIGINAL DBHANDLER ORIGINALLY PROVIDED BY MY COMPUTER SCIENCE TEACHER,* ***MR FROS****T. BOTH THE ORIGINAL AND MODIFIED SOURCE CODE ARE DISPLAYED BELOW.* | | |
| Original: | | |
| import mysql.connector  from mysql.connector import errorcode  class DBAccess:    \_\_username = "yourforename"  \_\_password = "yoursurname"  \_\_host = "10.105.2.40"  \_\_database = "yourforename"  connection = ""  DBcursor = ""    def \_\_init\_\_(self):  try:  self.connection = mysql.connector.connect(user=self.\_\_username,password=self.\_\_password,  host=self.\_\_host,  database=self.\_\_database)  self.DBcursor = self.connection.cursor(prepared=True)    except mysql.connector.Error as err:  if err.errno == errorcode.ER\_ACCESS\_DENIED\_ERROR:  print("Something is wrong with your user name or password")  elif err.errno == errorcode.ER\_BAD\_DB\_ERROR:  print("Database does not exist")  else:  print(err)  #else:  # connection.close()    def insert(self, sqlStatement, valuesList):  self.DBcursor.execute(sqlStatement, valuesList) # sending a prepared statement with a list of values in order  self.connection.commit() # confirm it is the correct operation    def close(self):  self.connection.close() | | |
| Modified: | | |
| #database imports  import mysql.connector  from mysql.connector import errorcode  #general imports  from dotenv import load\_dotenv  import os  #overarching parent class for handling database connections. All child classes will serve a specific function  class DBHandler:  #Load environment variables from the .env file  envVarPath="src/config/.env"  load\_dotenv(dotenv\_path=envVarPath)  \_\_username = os.getenv('DB\_USERNAME')  \_\_password = os.getenv('DB\_PASSWORD')  \_\_host = os.getenv('DB\_HOST')  \_\_schema = os.getenv('DB\_SCHEMA')  \_defAlertEmail = os.getenv('DEF\_ALERT\_RECIPIENT\_EMAIL')  connection = ""  cursor = ""  def \_\_init\_\_(self):  try:  self.connection = mysql.connector.connect(user=self.\_\_username, password=self.\_\_password, host=self.\_\_host, database=self.\_\_schema)  self.cursor = self.connection.cursor(prepared=True)  print(self.isConnected())  except mysql.connector.Error as err:  if err.errno == errorcode.ER\_ACCESS\_DENIED\_ERROR:  print(err)  print("Something is wrong with your user name or password")  elif err.errno == errorcode.ER\_BAD\_DB\_ERROR:  print("Database does not exist")  else:  print(err)  #method to test connection to server  def isConnected(self):  if self.connection.is\_connected():  return f"Database {self} initialised successfully"    return False  def getCount(self, tableName, displayType=True):  try:  self.cursor.execute(f"SELECT COUNT(\*) FROM {tableName}")  rowCount = self.cursor.fetchone()[0]  if displayType:  spacing = 3-len(str(rowCount))  zeros = spacing\*"0"  prettified = f"{zeros}{rowCount} {tableName}"  return prettified    else:  return rowCount  except Exception as error:  print(f"Error encountered (getCount): {error}")  return None    def getColumnNames(self, tableName):  self.cursor.execute(f"SELECT \* FROM {tableName} LIMIT 0")  self.cursor.fetchall()  columnNames = []  for description in self.cursor.description:  columnNames.append(description[0])  return columnNames    def getColumnData(self, columnName, tableName):  self.cursor.execute(f"SELECT {columnName} FROM {tableName}")    return self.cursor.fetchall()    def getColumnCount(self, tableName):  self.cursor.execute(f"SELECT \* FROM {tableName} LIMIT 1")  self.cursor.fetchall()  columnCount = len(self.cursor.description)    return columnCount  def getTables(self):  try:  self.cursor.execute("SHOW TABLES")  tables = self.cursor.fetchall()  tableNames = [table[0] for table in tables]  return tableNames  except Exception as error:  print(f"error encountered: {error}")  def getData(self, tableName):  self.cursor.execute(f"SELECT \* FROM {tableName}")  return self.cursor.fetchall()    def generalUpdateRecord(self, tableName, columnName, oldVar, newVar):  self.cursor.execute(f"INSERT INTO {tableName} ({columnName}) VALUES (%s) WHERE {columnName} == {oldVar}", (newVar,))  def close(self):  self.connection.close() | | |

|  |  |  |
| --- | --- | --- |
| logonDBHandler.py – logon handling class | Requirement(s) met: **S3, M10, M10A, M10B** | Technicality: Group B |
| #database imports  import mysql.connector  from mysql.connector import errorcode  #general imports  import hashlib  import string  import secrets  #import proccesses and any db handlers  from processes.popUpWindow import \*  from processes.sendEmail import \*  from dbHandling.DBHandler import \*  class logonDBHandler(DBHandler):  def initializeDatabase(self):  try:  self.cursor.execute('''  CREATE TABLE IF NOT EXISTS users (  user\_id INT auto\_increment PRIMARY KEY,  username VARCHAR(50) NOT NULL,  password VARCHAR(64) NOT NULL,  access\_level VARCHAR(50) NOT NULL,  recovery\_code VARCHAR(100) NOT NULL,  email\_address VARCHAR(100) NOT NULL  )  ''')  self.connection.commit()    except Exception as error:  return False, error  def createUserCreds(self, username, password, accessLevel, emailAddress):  if self.validateUser(username, password):  message = popUpWindow("User already exists")  message.create()  return False    else:  #create the account recovery code in case a user forgets their password and they need to reset it  recoveryCode = self.createAccRecoveryCode()  #create a pop-up window with the users recovery code as well as sending them an email containing it  message = popUpWindow(f"Recovery code: {recoveryCode}")  message.create()  newEmail = appEmail()  newEmail.sendEmail(emailAddress, "Recovery code for Onestop Stock Monitoring Assistant", f"Recovery code: {recoveryCode}")    try:  #Use parameterized query for safety  self.cursor.execute("""INSERT INTO users (username, password, access\_level, recovery\_code, email\_address) VALUES (%s, %s, %s, %s, %s)""",(username,logonDBHandler.hashData(str(password)),accessLevel,logonDBHandler.hashData(str(recoveryCode)),emailAddress,))  message = popUpWindow("User created successfully")  except Exception as e:  print(e)  self.connection.commit()  def readUserCreds(self):  self.cursor.execute('SELECT user\_id, username, password, access\_level FROM users')  rows = self.cursor.fetchall()  self.connection.close()    return rows    def getUserNames(self, current=False):  try:  self.cursor.execute("SELECT username FROM users")  results = self.cursor.fetchall()  print(results)  userNames = [row[0] for row in results]  return userNames    except Exception as error:  return False, error  def validateUser(self, providedUsername, providedPassword):  try:  hashedPassword = logonDBHandler.hashData(str(providedPassword))  self.cursor.execute('SELECT user\_id, access\_level FROM users WHERE username = %s AND password = %s',(providedUsername, hashedPassword))  data = self.cursor.fetchone()  print(f"Query Result: {data}") # Debugging output  return bool(data) # Return True if data is found, else False    except Exception as e:  print(f"Error during user validation: {e}")  return False    def changePasswordProcess(self, username, old\_password, new\_password):  self.cursor.execute("SELECT password FROM users WHERE username = %s", (username,))  current\_password = self.cursor.fetchall()  if current\_password is None:  return False  elif current\_password[0][0] == logonDBHandler.hashData(str(old\_password)):  self.cursor.execute('UPDATE users SET password = %s WHERE username = %s', (logonDBHandler.hashData(str(new\_password)), username))  self.connection.commit()    return True    else:  return False    def genTempPass(self):  tempPass = ""  for i in range (4):  tempPass += secrets.choice(string.digits)    return tempPass    def changePasswordOutright(self, accountName, newPassword):  self.cursor.execute('UPDATE users SET password = %s WHERE username = %s', (logonDBHandler.hashData(str(newPassword)), accountName))  self.connection.commit()  def createAccRecoveryCode(self):  recoveryCode = ""  for i in range(3):  recoveryCode += secrets.choice(string.ascii\_uppercase)  recoveryCode += "-"  for i in range(3):  recoveryCode += secrets.choice(string.digits)  return recoveryCode  def validateRecoveryCode(self, username, leftH, rightH):  self.cursor.execute('SELECT recovery\_code FROM users WHERE username = %s', (username,))  result = self.cursor.fetchone()[0]  recoveryCode = f"{leftH}-{rightH}"  if logonDBHandler.hashData(str(recoveryCode)) == result:  return True    return False    def getUserAccessLevel(self, username):  self.cursor.execute('SELECT access\_level FROM users WHERE username = %s', (username,))  result = self.cursor.fetchone()[0]  return result  #<=======================STATIC-METHODS=======================>#    def hashData(data):  return hashlib.sha256(str.encode(data)).hexdigest() | | |

|  |  |  |
| --- | --- | --- |
| productDBHandler.py – product database handling | Requirement(s) met: **M2, M2A, M2B** | Technicality: Group B |
| from dbHandling.DBHandler import \*  class productDBHandler(DBHandler):  def initializeDatabase(self):  try:  self.cursor.execute('''  CREATE TABLE IF NOT EXISTS products (  product\_id INT AUTO\_INCREMENT PRIMARY KEY,  supplier\_id INT,  product\_name VARCHAR(100) NOT NULL,  product\_description VARCHAR(200),  product\_pack\_size INT NOT NULL,  product\_weight INT NOT NULL,  product\_barcode VARCHAR(50) NOT NULL,  product\_price DECIMAL(10,2) NOT NULL,  FOREIGN KEY (supplier\_id) REFERENCES suppliers(supplier\_ID)  )  ''')  except Exception as error:  return False, error    def createProduct(self, supplier\_id, product\_name, product\_description, product\_pack\_size, product\_weight, product\_price, product\_barcode=000000000):  try:  self.cursor.execute('''INSERT INTO products (supplier\_id, product\_name, product\_description, product\_pack\_size, product\_weight, product\_barcode, product\_price) VALUES (%s, %s, %s, %s, %s, %s, %s)''', (supplier\_id, product\_name, product\_description, product\_pack\_size, product\_weight, product\_barcode, product\_price))  self.connection.commit()  return True    except Exception as error:  self.connection.rollback()  print(f"Error in create product: {error}")  return False, error    def getProductNames(self):  try:  self.cursor.execute("SELECT product\_name FROM products")  results = self.cursor.fetchall()  productNames = [row[0] for row in results]  return productNames    except Exception as error:  return False, error  def getProductID(self, productName):  try:  self.cursor.execute("SELECT product\_id FROM products WHERE product\_name = %s", (productName,))  results = self.cursor.fetchone()  return results[0]    except Exception as error:  return False, error    def getProductName(self, productID):  try:  self.cursor.execute("SELECT product\_name FROM products WHERE product\_id = %s", (productID,))  results = self.cursor.fetchone()  return results[0]    except Exception as error:  return False, error    def getRespectiveSupplerID(self, productName):  try:  self.cursor.execute("SELECT supplier\_id FROM products WHERE product\_name = %s", (productName,))  results = self.cursor.fetchone()  return results[0]    except Exception as error:  return False, error  #need to program  def deleteProduct(self):  pass  def updateProductValue(self, dbColumnVal, productID):  self.cursor.execute("UPDATE products SET %s WHERE product\_id = %s", (dbColumnVal, productID))  self.connection.commit() | | |

|  |  |  |
| --- | --- | --- |
| stockLevelDBHandler.py – stock level database handling | Requirement(s) met: **M1, M1B** | Technicality: Group B |
| from dbHandling.DBHandler import \*  from dbHandling.stockLevelHistoryDBHandler import \*  from dbHandling.productDBHandler import \*  import json  class stockLevelDBHandler(DBHandler):  stockLevelHistoryDB = stockLevelHistoryDBHandler()  productDBHandler\_ = productDBHandler()  def initializeDatabase(self):  try:  self.cursor.execute('''  CREATE TABLE IF NOT EXISTS stockLevel (  stock\_id INT AUTO\_INCREMENT PRIMARY KEY,  product\_id INT,  stock\_count INT NOT NULL,  minimum\_stock\_level INT NOT NULL,  reOrder\_level INT NOT NULL,  lastDelivery JSON,  FOREIGN KEY (product\_id) REFERENCES products(product\_ID)  )  ''')  except Exception as error:  return False, error  def addStockLevelData(self, productID, minStockLevel, reOrderLevel, \*, stockCount=0, lastDelivery="[]"):  try:  if isinstance(lastDelivery, dict):  lastDelivery = json.dumps(lastDelivery)  params = (productID, minStockLevel, reOrderLevel, stockCount, lastDelivery)  self.cursor.execute('''INSERT INTO stocklevel (product\_id, minimum\_stock\_level, reOrder\_level, stock\_count, lastDelivery) VALUES (%s, %s, %s, %s, %s)''', params)  self.connection.commit()  stockID = self.getStockID(productID)  self.stockLevelHistoryDB.addStockLevelHistoryData(stockID, productID, self.productDBHandler\_.getProductName(productID), stockCount)  return True  except Exception as error:  self.connection.rollback()  print(f"sldb: {error}")  return False, error    def updateStockLevel(self, addedStockCount, productID, isDelivery=False, isWaste=False):  try:  self.cursor.execute("SELECT stock\_count FROM stockLevel WHERE product\_id = %s", (productID,))  stockLevelNum = self.cursor.fetchone()  print(stockLevelNum, addedStockCount)  stockUpdateType = ""  if isDelivery: #if the update stock level is for a delivery, we are simply adding more not overwriting the previous stock level (as a delivery = more stock)  self.cursor.execute("UPDATE stockLevel SET stock\_count = stock\_count + %s WHERE product\_id = %s", (addedStockCount+stockLevelNum[0], productID))  stockUpdateType = "delivery"    elif isWaste:  self.cursor.execute("UPDATE stockLevel SET stock\_count = stock\_count - %s WHERE product\_id = %s", (addedStockCount, productID))  stockUpdateType = "waste"  else:  self.cursor.execute("UPDATE stockLevel SET stock\_count = %s WHERE product\_id = %s", (addedStockCount, productID))  stockUpdateType = "count"  self.connection.commit()  #update stock level history table with change in stockLevel  stockID = self.getStockID(productID)  self.stockLevelHistoryDB.addStockLevelHistoryData(stockID, productID, self.productDBHandler\_.getProductName(productID), addedStockCount, stockUpdateType)  except Exception as error:  print(f"error in updateStockLevel: {error}")  return False    def getStockID(self, productID):  try:  self.cursor.execute("SELECT stock\_Id FROM stockLevel WHERE product\_id = %s", (productID,))  return self.cursor.fetchone()[0]  except Exception as error:  print(f"error in getStockId: {error}")  return False   def updateLastDelivery(self, lastDelivery, productID):          try:              self.cursor.execute("UPDATE stockLevel SET lastDelivery = %s WHERE product\_id = %s", (lastDelivery,productID))              self.connection.commit()          except Exception as error:              print(f"error: {error}")              return False | | |

|  |  |  |
| --- | --- | --- |
| stockLevelHistoryDBHandler.py – stock level history database handler | Requirement(s) met: **M1, M1E** | Technicality: Group B |
| from dbHandling.DBHandler import \*  class stockLevelHistoryDBHandler(DBHandler):  def initializeDatabase(self):  try:  self.cursor.execute('''  CREATE TABLE IF NOT EXISTS stockLevelHistory (  stockHistory\_id INT AUTO\_INCREMENT PRIMARY KEY,  stock\_id INT,  product\_id INT,  stock\_history\_product\_name VARCHAR(100) NOT NULL,  stock\_count INT,  action VARCHAR(100) NOT NULL,  date DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,  FOREIGN KEY (stock\_id) REFERENCES stocklevel(stock\_id),  FOREIGN KEY (product\_id) REFERENCES products(product\_id)  )  ''')  except Exception as error:  return False, error    def addStockLevelHistoryData(self, stockID, productID, stockHistoryProductName, stockCount, action):  try:  params = (stockID, productID, stockHistoryProductName, stockCount, action)  self.cursor.execute('''INSERT INTO stockLevelHistory (stock\_id, product\_id, stock\_history\_product\_name, stock\_count, action) VALUES (%s, %s, %s, %s, %s)''', params)  self.connection.commit()  return True  except Exception as error:  self.connection.rollback()  print(f"slhdb: {error}")  return False, error    def getGraphValues(self, productName):  try:  self.cursor.execute('''SELECT date, stock\_count FROM stocklevelhistory WHERE stock\_history\_product\_name = %s''', (productName,))  data = self.cursor.fetchall()  return data  except Exception as error:  self.connection.rollback()  print(f"slhdb: {error}")  return False, error | | |

|  |  |  |
| --- | --- | --- |
| supplierDBHandler.py – supplier database handling | Requirement(s) met: **M3, M3A, M3B** | Technicality: Group B |
| from dbHandling.DBHandler import \*  class supplierDBHandler(DBHandler):  def initializeDatabase(self):  try:  self.cursor.execute('''  CREATE TABLE IF NOT EXISTS suppliers (  supplier\_id INT auto\_increment PRIMARY KEY,  supplier\_name VARCHAR(100) NOT NULL,  supplier\_description CHAR(255) NOT NULL,  supplier\_delivery\_date JSON  )  ''')  except Exception as error:  return False, error    def getSupplierNames(self):  try:  self.cursor.execute("SELECT supplier\_name FROM suppliers")  results = self.cursor.fetchall()  supplierNames = [row[0] for row in results]    return supplierNames    except Exception as error:  return False, error    def createSupplier(self, supplier\_name, supplier\_description, supplier\_delivery\_date):  try:  self.cursor.execute('''INSERT INTO suppliers (supplier\_name, supplier\_description, supplier\_delivery\_date) VALUES (%s, %s, %s)''', (supplier\_name, supplier\_description, supplier\_delivery\_date))  self.connection.commit()  return True    except Exception as error:  self.connection.rollback()  return False, error      def getSupplierID(self, supplier\_name):  try:  self.cursor.execute('''SELECT supplier\_id FROM suppliers WHERE supplier\_name = %s''', (supplier\_name,))  result = self.cursor.fetchone()  return result    except Exception as error:  self.connection.rollback()  return False, error  #needs to be programmed  def deleteSupplier(self):  pass  def getSupplierDeliveryDates(self):  pass | | |

|  |  |  |
| --- | --- | --- |
| wasteDBHandler.py – waste database handling | Requirement(s) met: **M6** | Technicality: Group B |
| from dbHandling.DBHandler import \*  class wasteDBHandler(DBHandler):  def initializeDatabase(self):  try:  self.cursor.execute('''  CREATE TABLE IF NOT EXISTS waste (  waste\_id INT AUTO\_INCREMENT PRIMARY KEY,  product\_id INT,  supplier\_id INT,  waste\_reason VARCHAR(200) NOT NULL,  waste\_dealt\_with BOOLEAN NOT NULL,  FOREIGN KEY (product\_id) REFERENCES products(product\_id),  FOREIGN KEY (supplier\_id) REFERENCES suppliers(supplier\_id)  )  ''')  except Exception as error:  return False, error    def createWasteProduct(self, product\_id, supplier\_id, waste\_reason, waste\_dealt\_with):  try:  self.cursor.execute('''INSERT INTO waste (product\_id, supplier\_id, waste\_reason, waste\_dealt\_with) VALUES (%s, %s, %s, %s)''', (product\_id, supplier\_id, waste\_reason, waste\_dealt\_with))  self.connection.commit()  return True    except Exception as error:  self.connection.rollback()  print(f"Error in create waste product: {error}")  return False, error | | |

|  |  |  |
| --- | --- | --- |
| weeklyReportRecordDBHandler.py – weekly report records table database handler | Requirement(s) met: **M9** | Technicality: A |
| from dbHandling.DBHandler import \*  import json  class weeklyReportDBHandler(DBHandler):  def initializeDatabase(self):  try:  self.cursor.execute('''  CREATE TABLE IF NOT EXISTS weeklyReportRecords (  weekly\_report\_record\_id INT AUTO\_INCREMENT PRIMARY KEY,  product\_id INT,  weekly\_report\_generation\_date DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,  trend VARCHAR(100) NOT NULL,  predicted JSON,  revenue DECIMAL(10,2) NOT NULL,  cost\_of\_goods\_sold DECIMAL(10,2) NOT NULL,  net\_profit DECIMAL(10,2) NOT NULL,  FOREIGN KEY (product\_id) REFERENCES products(product\_ID)  )  ''')  except Exception as error:  return False, error      def addWeeklyReportRecord(self, productID, trend, predicted, revenue, cogs, netProfit):  try:  if isinstance(predicted, dict):  predicted = json.dumps(predicted)  params = (productID, trend, predicted, revenue, cogs, netProfit)  self.cursor.execute('''INSERT INTO weeklyreportrecords (product\_id, trend, predicted, revenue, cost\_of\_goods\_sold, net\_profit) VALUES (%s, %s, %s, %s, %s, %s)''', params)  self.connection.commit()  except Exception as error:  self.connection.rollback()  print(f"sldb: {error}")  return False, error    def getWeeklyReportsAsList(self):  try:  self.cursor.execute("SELECT \* FROM weeklyreportrecords")  dataToUse = self.cursor.fetchall()  self.connection.commit()  groupedRecordIDs = []  for record in dataToUse:  if len(groupedRecordIDs) == 0:  groupedRecordIDs.append([record[2].strftime("%d/%m/%Y"), record[0]])  else:  count1 = 0  for prevRecord in groupedRecordIDs:  if record[2].strftime("%d/%m/%Y") == prevRecord[0]:  count1 += 1  prevRecord.append(record[0])    if count1 == 0:  groupedRecordIDs.append([record[2].strftime("%d/%m/%Y"), record[0]])  return groupedRecordIDs, dataToUse  except Exception as error:  self.connection.rollback()  print(f"wrdb: {error}")  return False, error | | |

*icos-folder source*

|  |  |  |
| --- | --- | --- |
| person.ico – person icon | Requirement(s) met: n/a | Technicality: n/a |
|  | | |

*processes-folder source*

|  |  |  |
| --- | --- | --- |
| autoCompleteSearch.py – autoComplete search object | Requirement(s) met: **S7, S7A** | Technicality: Group B |
| import customtkinter as ctk  import tkinter as tk  class AutocompleteEntry(ctk.CTkEntry):  def \_\_init\_\_(self, master=None, \*\*kwargs):  super().\_\_init\_\_(master, \*\*kwargs)  self.suggestionsButtons = [] #list to store the suggestion buttons  self.suggestions = []  self.allSuggestions = []  self.bind("<KeyRelease>", self.onKeyrelease)  def onKeyrelease(self, event):  typed = self.get().lower()  if typed == "":  self.hideSuggestions()  return  self.suggestions = [s for s in self.allSuggestions if typed in s.lower()]    if self.suggestions:  self.showSuggestions()  else:  self.hideSuggestions()  def getEntryData(self):  return self.get()  def showSuggestions(self):  self.hideSuggestions()  for i, suggestion in enumerate(self.suggestions):  button = ctk.CTkButton(self.master, text=suggestion, command=lambda s=suggestion: self.onSuggestionClick(s))  button.place(x=self.winfo\_x(), y=self.winfo\_y() + self.winfo\_height() + (i \* 30), anchor="w")  self.suggestionsButtons.append(button)  def hideSuggestions(self):  for button in self.suggestionsButtons:  button.destroy()  self.suggestionsButtons.clear()  def onSuggestionClick(self, suggestion):  self.delete(0, tk.END)  self.insert(0, suggestion)  self.hideSuggestions()  def setSuggestions(self, suggestions):  self.allSuggestions = suggestions | | |

|  |  |  |
| --- | --- | --- |
| changePassword.py – change password UI & functionality | Requirement(s) met: **M10, M10A** | Technicality: Group B |
| import customtkinter  from dbHandling.logonDBHandler import \*  from processes.forgotPassword import \*  from processes.popUpWindow import \*  from processes.windowSuperClass import superWindow  customtkinter.set\_default\_color\_theme("dark-blue")  class changePassword(superWindow):        APP\_NAME = "Change Password"      WIDTH = 500      HEIGHT = 250        def \_\_init\_\_(self, overWrite, \*args, \*\*kwargs):          super().\_\_init\_\_(\*args, \*\*kwargs)          self.overwrite = overWrite          self.title(changePassword.APP\_NAME)          self.geometry(str(changePassword.WIDTH) + "x" + str(changePassword.HEIGHT))          self.minsize(changePassword.WIDTH, changePassword.HEIGHT)            #create change password frame          self.changePasswordFrame = customtkinter.CTkFrame(self, corner\_radius=10)          self.changePasswordFrame.place(relx=0.5, rely=0.5, anchor=customtkinter.CENTER)            self.usernameLabel = customtkinter.CTkLabel(self.changePasswordFrame, text="Enter Username:", anchor="w")          self.usernameLabel.grid(row=0, column=0)            self.usernameEntry = customtkinter.CTkEntry(self.changePasswordFrame, placeholder\_text="Username...")          self.usernameEntry.grid(row=0, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.oldPasswordLabel = customtkinter.CTkLabel(self.changePasswordFrame, text="Enter Old Password:", anchor="w")          self.oldPasswordLabel.grid(row=1, column=0)            self.oldPasswordEntry = customtkinter.CTkEntry(self.changePasswordFrame, show="\*", placeholder\_text="Old Password...")          self.oldPasswordEntry.grid(row=1, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.newPasswordLabel = customtkinter.CTkLabel(self.changePasswordFrame, text="Enter New Password:", anchor="w")          self.newPasswordLabel.grid(row=2, column=0)            self.newPasswordEntry = customtkinter.CTkEntry(self.changePasswordFrame, show="\*", placeholder\_text="New Password...")          self.newPasswordEntry.grid(row=2, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.buttonChangePassword = customtkinter.CTkButton(self.changePasswordFrame, text="Change", command=self.change)          self.buttonChangePassword.grid(row=3, column=0, sticky="w", padx=(12, 0), pady=12)            self.buttonForgotPassword = customtkinter.CTkButton(self.changePasswordFrame, text="Forgot password", command=self.forgotPassword)          self.buttonForgotPassword.grid(row=3, column=1, sticky="w", padx=(12, 12), pady=12)            self.buttonExit = customtkinter.CTkButton(self.changePasswordFrame, text="Exit", command=self.onClosing)          self.buttonExit.grid(row=3, column=2, sticky="w", padx=(0, 12), pady=12)        def change(self):          logon\_ = logonDBHandler()          logon\_.initializeDatabase()            print(self.usernameEntry.get(), self.oldPasswordEntry.get(), self.newPasswordEntry.get())          changePass = logon\_.changePasswordProcess(self.usernameEntry.get(), self.oldPasswordEntry.get(), self.newPasswordEntry.get(), self.overwrite)            if changePass:              print("Password Changed")              self.on\_closing()              message = popUpWindow("Password changed successfuly")              message.create()            else:              self.usernameEntry.configure(text\_color="red")              self.oldPasswordEntry.configure(text\_color="red")              message = popUpWindow("Incorrect information")              message.create()        def forgotPassword(self):          forgotPasswordWin = forgotPassword()          forgotPasswordWin.mainloop() | | |

|  |  |  |
| --- | --- | --- |
| CTkDatePicker.py – python library source code | Requirement(s) met: **S2, S2A** | Technicality: n/a |
| *AS MENTIONED PREVIOUSLY IN THIS DOCUMENTATION, “CTkDatePicker.py” WAS NOR WRITTEN OR CLAIMED TO HAVE BEEN WRITTEN BY ME. THIS IS CLEARLY DETAILED IN* [*PYTHON LIBRARY/PACKAGE USAGE TABLE OF REFERENCES*](#_Python_library/package_usage) *ON PAGE 15 INCLUDING WHY THE SOURCE CODE NEEDED TO BE MANUALLY DOWNLOADED INSTEAD OF IMPORTING NORMALLY VIA PIP.* | | |
| """  CTkDatePicker is a custom date picker widget built using the CustomTkinter library. It provides a user-friendly interface for selecting dates, with both a text entry and a calendar popup for easy date selection.  Link: https://github.com/maxverwiebe/CTkDatePicker/tree/main  Author: Max Verwiebe  """  import tkinter as tk  import customtkinter as ctk  from datetime import datetime  import calendar  class CTkDatePicker(ctk.CTkFrame):  def \_\_init\_\_(self, master=None, \*\*kwargs):  """  Initialize the CTkDatePicker instance.    Parameters:  - master: The parent widget.  - \*\*kwargs: Additional keyword arguments passed to the CTkFrame constructor.    Initializes the date entry, calendar button, popup, and other related components.  """    super().\_\_init\_\_(master, \*\*kwargs)  self.date\_entry = ctk.CTkEntry(self, placeholder\_text="xx/xx/xx...")  self.date\_entry.grid(row=0, column=0, sticky="ew", padx=5, pady=5)    self.calendar\_button = ctk.CTkButton(self, text="▼", width=20, command=self.open\_calendar)  self.calendar\_button.grid(row=0, column=1, sticky="ew", padx=5, pady=5)  self.popup = None  self.selected\_date = None  self.date\_format = "%d/%m/%Y"  self.allow\_manual\_input = True    def set\_date\_format(self, date\_format):  """  Set the date format to be used in the date entry.  Parameters:  - date\_format (str): The desired date format string, e.g., "%m/%d/%Y".    Sets the format in which the selected date will be displayed.  """  self.date\_format = date\_format    def open\_calendar(self):  """  Open the calendar popup for date selection.    Creates and displays a calendar widget allowing the user to select a date.  The calendar appears just below the date entry field.  """    if self.popup is not None:  self.popup.destroy()  self.popup = ctk.CTkToplevel(self)  self.popup.title("Select Date")  self.popup.geometry("+%d+%d" % (self.winfo\_rootx(), self.winfo\_rooty() + self.winfo\_height()))  self.popup.resizable(False, False)    self.popup.after(500, lambda: self.popup.focus())    self.current\_year = datetime.now().year  self.current\_month = datetime.now().month  self.build\_calendar()    def build\_calendar(self):  """  Build and display the calendar in the popup.  Constructs the calendar UI for the currently selected month and year.  Includes navigation buttons for previous and next months.  """    if hasattr(self, 'calendar\_frame'):  self.calendar\_frame.destroy()  self.calendar\_frame = ctk.CTkFrame(self.popup)  self.calendar\_frame.grid(row=0, column=0)  # Month and Year Selector  month\_label = ctk.CTkLabel(self.calendar\_frame, text=f"{calendar.month\_name[self.current\_month]}, {self.current\_year}")  month\_label.grid(row=0, column=1, columnspan=5)  prev\_month\_button = ctk.CTkButton(self.calendar\_frame, text="<", width=5, command=self.prev\_month)  prev\_month\_button.grid(row=0, column=0)  next\_month\_button = ctk.CTkButton(self.calendar\_frame, text=">", width=5, command=self.next\_month)  next\_month\_button.grid(row=0, column=6)  # Days of the week header  days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]  for i, day in enumerate(days):  lbl = ctk.CTkLabel(self.calendar\_frame, text=day)  lbl.grid(row=1, column=i)  # Days in month  month\_days = calendar.monthrange(self.current\_year, self.current\_month)[1]  start\_day = calendar.monthrange(self.current\_year, self.current\_month)[0]  day = 1  for week in range(2, 8):  for day\_col in range(7):  if week == 2 and day\_col < start\_day:  lbl = ctk.CTkLabel(self.calendar\_frame, text="")  lbl.grid(row=week, column=day\_col)  elif day > month\_days:  lbl = ctk.CTkLabel(self.calendar\_frame, text="")  lbl.grid(row=week, column=day\_col)  else:  btn = ctk.CTkButton(self.calendar\_frame, text=str(day), width=3, command=lambda day=day: self.select\_date(day), fg\_color="transparent")  btn.grid(row=week, column=day\_col)  day += 1  def prev\_month(self):  """  Navigate to the previous month in the calendar.  Updates the calendar display to show the previous month's days.  Adjusts the year if necessary.  """    if self.current\_month == 1:  self.current\_month = 12  self.current\_year -= 1  else:  self.current\_month -= 1  self.build\_calendar()  def next\_month(self):  """  Navigate to the next month in the calendar.  Updates the calendar display to show the next month's days.  Adjusts the year if necessary.  """    if self.current\_month == 12:  self.current\_month = 1  self.current\_year += 1  else:  self.current\_month += 1  self.build\_calendar()  def select\_date(self, day):  """  Select a date from the calendar.  Parameters:  - day (int): The day of the month selected by the user.    Sets the selected date in the date entry field and closes the calendar popup.  """    self.selected\_date = datetime(self.current\_year, self.current\_month, day)  # Temporarily enable the entry to set the date  self.date\_entry.configure(state='normal')  self.date\_entry.delete(0, tk.END)  self.date\_entry.insert(0, self.selected\_date.strftime(self.date\_format))  # Restore the disabled state if necessary  if not self.allow\_manual\_input:  self.date\_entry.configure(state='disabled')  self.popup.destroy()  self.popup = None    def get\_date(self):  """  Get the currently selected date as a string.  Returns:  - str: The date string in the format specified by self.date\_format.  """    return self.date\_entry.get()    def set\_allow\_manual\_input(self, value):  """  Enable or disable manual date input.  Parameters:  - value (bool): If True, manual input in the date entry is allowed; otherwise, it is disabled.    Allows the user to manually enter a date if set to True; otherwise, restricts input to selection via the calendar.  """    self.allow\_manual\_input = value  if not value:  self.date\_entry.configure(state='disabled')  else:  self.date\_entry.configure(state='normal') | | |

|  |  |  |
| --- | --- | --- |
| doubleAxesScrollingFrame.py – python library source code | Requirement(s) met: **S2, S2A** | Technicality: n/a |
| *AS MENTIONED PREVIOUSLY IN THIS DOCUMENTATION, “*doubleAxesScrollingFrame*.py” WAS NOR WRITTEN OR CLAIMED TO HAVE BEEN WRITTEN BY ME. THIS IS CLEARLY DETAILED IN* [*PYTHON LIBRARY/PACKAGE USAGE TABLE OF REFERENCES*](#_Python_library/package_usage) *ON PAGE 15 INCLUDING WHY THE SOURCE CODE NEEDED TO BE MANUALLY DOWNLOADED INSTEAD OF IMPORTING NORMALLY VIA PIP.* | | |
| """  Advanced scrollable frame for customtkinter  Author: Akash Bora  License: MIT  Link: https://github.com/Akascape/CTkXYFrame  """  import customtkinter  from tkinter import Canvas  class CTkXYFrame(customtkinter.CTkFrame):  def \_\_init\_\_(self,  master: any,  width: int = 100,  height: int = 100,  scrollbar\_width: int = 16,  scrollbar\_fg\_color = None,  scrollbar\_button\_color = None,  scrollbar\_button\_hover\_color = None,  \*\*kwargs):  self.parent\_frame = customtkinter.CTkFrame(master=master, \*\*kwargs)  self.bg\_color = self.parent\_frame.cget("fg\_color")  self.xy\_canvas = Canvas(self.parent\_frame, width=width, height=height,  bg=self.parent\_frame.\_apply\_appearance\_mode(self.bg\_color),  borderwidth=0, highlightthickness=0)  self.parent\_frame.rowconfigure(0,weight=1)  self.parent\_frame.columnconfigure(0,weight=1)    customtkinter.CTkFrame.\_\_init\_\_(self, master=self.xy\_canvas, fg\_color=self.parent\_frame.cget("fg\_color"),  bg\_color=self.parent\_frame.cget("fg\_color"))  self.window\_id = self.xy\_canvas.create\_window((0,0), window=self, anchor="nw")    self.vsb = customtkinter.CTkScrollbar(self.parent\_frame, orientation="vertical", command=self.xy\_canvas.yview,  fg\_color=scrollbar\_fg\_color, button\_color=scrollbar\_button\_color,  button\_hover\_color=scrollbar\_button\_hover\_color, width=scrollbar\_width)  self.hsb = customtkinter.CTkScrollbar(self.parent\_frame, orientation="horizontal", command=self.xy\_canvas.xview,  fg\_color=scrollbar\_fg\_color, button\_color=scrollbar\_button\_color,  button\_hover\_color=scrollbar\_button\_hover\_color, height=scrollbar\_width)    self.xy\_canvas.configure(yscrollcommand=lambda x,y: self.dynamic\_scrollbar\_vsb(x,y),  xscrollcommand=lambda x,y: self.dynamic\_scrollbar\_hsb(x,y))  self.xy\_canvas.grid(row=0, column=0, sticky="nsew", padx=(7,0), pady=(7,0))    self.bind("<Configure>", lambda event, canvas=self.xy\_canvas: self.onFrameConfigure(canvas))  self.xy\_canvas.bind\_all("<MouseWheel>", lambda e: self.\_on\_mousewheel(e.delta, e.widget), add="+")  self.xy\_canvas.bind\_all("<Shift-MouseWheel>", lambda e: self.\_on\_mousewheel\_shift(e.delta, e.widget), add="+")  self.xy\_canvas.bind\_all("<Button-4>", lambda e: self.\_on\_mousewheel(120, e.widget), add="+")  self.xy\_canvas.bind\_all("<Button-5>", lambda e: self.\_on\_mousewheel(-120, e.widget), add="+")  self.xy\_canvas.bind\_all("<Shift-Button-4>", lambda e: self.\_on\_mousewheel\_shift(120, e.widget), add="+")  self.xy\_canvas.bind\_all("<Shift-Button-5>", lambda e: self.\_on\_mousewheel\_shift(-120, e.widget), add="+")  if type(master) is customtkinter.CTkScrollableFrame:  master.check\_if\_master\_is\_canvas = self.disable\_contentscroll    def destroy(self):  customtkinter.CTkFrame.destroy(self)  self.parent\_frame.destroy()  def \_set\_appearance\_mode(self, mode\_string):  super().\_set\_appearance\_mode(mode\_string)  self.xy\_canvas.config(bg=self.parent\_frame.\_apply\_appearance\_mode(self.bg\_color))  def check\_if\_master\_is\_canvas(self, widget):  if widget == self.xy\_canvas:  return True  elif widget.master is not None:  return self.check\_if\_master\_is\_canvas(widget.master)  else:  return False    def disable\_contentscroll(self, widget):  if widget == self.xy\_canvas:  return True  else:  return False    def dynamic\_scrollbar\_vsb(self, x, y):  if float(x)==0.0 and float(y)==1.0:  self.vsb.grid\_forget()  else:  self.vsb.grid(row=0, column=1, rowspan=2, sticky="nse", pady=5)  self.vsb.set(x,y)    def dynamic\_scrollbar\_hsb(self, x, y):  if float(x)==0.0 and float(y)==1.0:  self.hsb.grid\_forget()  else:  self.hsb.grid(row=1, column=0, sticky="nwe", padx=(5,0))  self.hsb.set(x,y)    def onFrameConfigure(self, canvas):  canvas.configure(scrollregion=canvas.bbox("all"))    def \_on\_mousewheel(self, event, widget):  if self.check\_if\_master\_is\_canvas(widget):  self.xy\_canvas.yview\_scroll(int(-1\*(event/120)), "units")    def \_on\_mousewheel\_shift(self, event, widget):  if self.check\_if\_master\_is\_canvas(widget):  self.xy\_canvas.xview\_scroll(int(-1\*(event/120)), "units")    def pack(self, \*\*kwargs):  self.parent\_frame.pack(\*\*kwargs)  def place(self, \*\*kwargs):  self.parent\_frame.place(\*\*kwargs)  def grid(self, \*\*kwargs):  self.parent\_frame.grid(\*\*kwargs)  def pack\_forget(self):  self.parent\_frame.pack\_forget()  def place\_forget(self, \*\*kwargs):  self.parent\_frame.place\_forget()  def grid\_forget(self, \*\*kwargs):  self.parent\_frame.grid\_forget()  def grid\_remove(self, \*\*kwargs):  self.parent\_frame.grid\_remove()  def grid\_propagate(self, \*\*kwargs):  self.parent\_frame.grid\_propagate()  def grid\_info(self, \*\*kwargs):  return self.parent\_frame.grid\_info()  def lift(self, aboveThis=None):  self.parent\_frame.lift(aboveThis)  def lower(self, belowThis=None):  self.parent\_frame.lower(belowThis)    def configure(self, \*\*kwargs):  if "fg\_color" in kwargs:  self.bg\_color = kwargs["fg\_color"]  self.xy\_canvas.config(bg=self.bg\_color)  self.configure(fg\_color=self.bg\_color)  if "width" in kwargs:  self.xy\_canvas.config(width=kwargs["width"])  if "height" in kwargs:  self.xy\_canvas.config(height=kwargs["height"])  self.parent\_frame.configure(\*\*kwargs) | | |

|  |  |  |
| --- | --- | --- |
| forgotPassword.py – forgot password UI & functionality | Requirement(s) met: **M10, M10C** | Technicality: B |
| import customtkinter  from dbHandling.logonDBHandler import \*  from processes.changePassword import \*  from processes.windowSuperClass import superWindow  customtkinter.set\_default\_color\_theme("dark-blue")  class forgotPassword(superWindow):        APP\_NAME = "Forgot Password"      WIDTH = 500      HEIGHT = 250        def \_\_init\_\_(self, \*args, \*\*kwargs):          super().\_\_init\_\_(\*args, \*\*kwargs)          self.title(forgotPassword.APP\_NAME)          self.geometry(str(forgotPassword.WIDTH) + "x" + str(forgotPassword.HEIGHT))          self.minsize(forgotPassword.WIDTH, forgotPassword.HEIGHT)            #create change password frame          self.forgotPasswordFrame = customtkinter.CTkFrame(self, corner\_radius=10)          self.forgotPasswordFrame.place(relx=0.5, rely=0.5, anchor=customtkinter.CENTER)            self.usernameLabel = customtkinter.CTkLabel(self.forgotPasswordFrame, text="Enter Username:", anchor="w")          self.usernameLabel.grid(row=0, column=0)            self.userEntry = customtkinter.CTkEntry(self.forgotPasswordFrame, placeholder\_text="Username...")          self.userEntry.grid(row=0, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.leftHandRCLabel = customtkinter.CTkLabel(self.forgotPasswordFrame, text="Left-hand R.C.:", anchor="w")          self.leftHandRCLabel.grid(row=1, column=0)            self.leftHandRCEntry = customtkinter.CTkEntry(self.forgotPasswordFrame, placeholder\_text="Recovery code left-hand...")          self.leftHandRCEntry.grid(row=1, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.rightHandRCLabel = customtkinter.CTkLabel(self.forgotPasswordFrame, text="Right-hand R.C.:", anchor="w")          self.rightHandRCLabel.grid(row=2, column=0)            self.rightHandRCEntry = customtkinter.CTkEntry(self.forgotPasswordFrame, placeholder\_text="Recovery code right-hand...")          self.rightHandRCEntry.grid(row=2, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.buttonValidate = customtkinter.CTkButton(self.forgotPasswordFrame, text="Validate R.C.", command=self.validateRecoveryPassword)          self.buttonValidate.grid(row=3, column=0, sticky="w", padx=(12, 0), pady=12)            self.buttonExit = customtkinter.CTkButton(self.forgotPasswordFrame, text="Exit", command=self.onClosing)          self.buttonExit.grid(row=3, column=2, sticky="w", padx=(12, 165), pady=12)        def validateRecoveryPassword(self):          check = logonDBHandler()          check.initializeDatabase()            if check.validateRecoveryCode(self.userEntry.get(), self.leftHandRCEntry.get(), self.rightHandRCEntry.get()):              tempPass = check.genTempPass()              print(f"Temporary password: {tempPass} - use this password to create a new password")              message = popUpWindow(f"Temporary password: {tempPass}")              message.create()              self.on\_closing()              # you now need to generate a new recovery code. this code is not displayed to the user, rather emailed, as a form of security.              changePasswordWin = changePassword(True)              changePasswordWin.mainloop()          else:              self.userEntry.configure(text\_color="red")              self.leftHandRCEntry.configure(text\_color="red")              self.rightHandRCEntry.configure(text\_color="red")              message = popUpWindow("Incorrect information")              message.create()  if \_\_name\_\_ == "\_\_main\_\_":      test = forgotPassword()      test.mainloop() | | |

|  |  |  |
| --- | --- | --- |
| loginProcess.py – login process UI & functionality | Requirement(s) met: **M10, M10A, M10B** | Technicality: B |
| import customtkinter  from dbHandling.logonDBHandler import \*  from processes.changePassword import \*  from processes.windowSuperClass import superWindow  from mainApp import \*  from time import sleep  from processes.stockLevelChecker import CheckStockCount  customtkinter.set\_default\_color\_theme("dark-blue")  class Logon(superWindow):        APP\_NAME = "Login Window"      WIDTH = 500      HEIGHT = 200        def \_\_init\_\_(self, \*args, \*\*kwargs):          super().\_\_init\_\_(\*args, \*\*kwargs)          self.title(Logon.APP\_NAME)          self.geometry(str(Logon.WIDTH) + "x" + str(Logon.HEIGHT))          self.minsize(Logon.WIDTH, Logon.HEIGHT)            #create login frame          self.loginFrame = customtkinter.CTkFrame(self, corner\_radius=10)          self.loginFrame.place(relx=0.5, rely=0.5, anchor=customtkinter.CENTER)            #create logon buttons and entries          self.usernameLabel = customtkinter.CTkLabel(self.loginFrame, text="Enter Username:", anchor="w")          self.usernameLabel.grid(row=0, column=0)          self.usernameEntry = customtkinter.CTkEntry(self.loginFrame, placeholder\_text="Username...")          self.usernameEntry.grid(row=0, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.passwordLabel = customtkinter.CTkLabel(self.loginFrame, text="Enter Password:", anchor="w")          self.passwordLabel.grid(row=1, column=0)          self.passwordEntry = customtkinter.CTkEntry(self.loginFrame, show="\*", placeholder\_text="Password...")          self.passwordEntry.grid(row=1, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)            self.buttonLogon = customtkinter.CTkButton(self.loginFrame, text="Login", command=self.logonProcess)          self.buttonLogon.grid(row=2, column=0, sticky="w", padx=(12, 0), pady=12)            self.buttonChangePassword = customtkinter.CTkButton(self.loginFrame, text="Change password", command=self.changePasswordWindow)          self.buttonChangePassword.grid(row=2, column=1, sticky="w", padx=(12, 12), pady=12)            self.buttonExit = customtkinter.CTkButton(self.loginFrame, text="Exit", command=self.onClosing)          self.buttonExit.grid(row=2, column=2, sticky="w", padx=(0, 12), pady=12)      def logonProcess(self):          self.logon\_ = logonDBHandler()          self.logon\_.initializeDatabase()          print(self.usernameEntry.get(), self.passwordEntry.get())          login = self.logon\_.validateUser(self.usernameEntry.get(), self.passwordEntry.get())          if login:              print("Hoorah")              self.newWindow()            else:              self.usernameEntry.configure(text\_color="red")              self.passwordEntry.configure(text\_color="red")              message = popUpWindow("Incorrect username or password")              message.create()      def changePasswordWindow(self):          changePasswordWin = changePassword(False)          changePasswordWin.mainloop()        def newWindow(self):          givenLevel = str(self)          givenUsername = self.usernameEntry.get()          self.onClosing()          message = popUpWindow("You have successfully logged in")          message.create()          app = App(givenLevel, givenUsername)          app.mainloop()      def \_\_str\_\_(self):          return self.logon\_.getUserAccessLevel(self.usernameEntry.get()) | | |

|  |  |  |
| --- | --- | --- |
| newUser.py – new user UI & functionality | Requirement(s) met: **M10, M10B, M10D, M12A** | Technicality: B |
| import customtkinter  from dbHandling.logonDBHandler import \*  from processes.changePassword import \*  from processes.windowSuperClass import superWindow  customtkinter.set\_default\_color\_theme("dark-blue")  class newUser(customtkinter.CTkToplevel):    APP\_NAME = "New User window"  WIDTH = 650  HEIGHT = 325    def \_\_init\_\_(self, \*args, \*\*kwargs):  super().\_\_init\_\_(\*args, \*\*kwargs)  self.title(newUser.APP\_NAME)  self.geometry(str(newUser.WIDTH) + "x" + str(newUser.HEIGHT))  self.minsize(newUser.WIDTH, newUser.HEIGHT)    #create new user frame  self.newUserFrame = customtkinter.CTkFrame(self, corner\_radius=10, width=750)  self.newUserFrame.place(relx=0.5, rely=0.5, anchor=customtkinter.CENTER)  #create widgets  self.usernameLabel = customtkinter.CTkLabel(self.newUserFrame, text="New Username:", anchor="w")  self.usernameLabel.grid(row=0, column=0)  self.usernameEntry = customtkinter.CTkEntry(self.newUserFrame, placeholder\_text="new username...")  self.usernameEntry.grid(row=0, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)    self.passwordLabel = customtkinter.CTkLabel(self.newUserFrame, text="New Password:", anchor="w")  self.passwordLabel.grid(row=1, column=0)  self.passwordEntry = customtkinter.CTkEntry(self.newUserFrame, show="\*", placeholder\_text="new password...")  self.passwordEntry.grid(row=1, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)  self.confirmpasswordLabel = customtkinter.CTkLabel(self.newUserFrame, text="Confirm password:", anchor="w")  self.confirmpasswordLabel.grid(row=1, column=3)  self.confirmpasswordEntry = customtkinter.CTkEntry(self.newUserFrame, show="\*", placeholder\_text="confirm password...")  self.confirmpasswordEntry.grid(row=1, column=4, columnspan=2, sticky="nsew", padx=(12), pady=12)  self.accessLevelLabel = customtkinter.CTkLabel(self.newUserFrame, text="Access level:", anchor="w")  self.accessLevelLabel.grid(row=2, column=0)  self.accessLevelEntry = customtkinter.CTkEntry(self.newUserFrame, placeholder\_text="access level...")  self.accessLevelEntry.grid(row=2, column=1, columnspan=2, sticky="nsew", padx=(12), pady=12)  self.emailAddrLabel = customtkinter.CTkLabel(self.newUserFrame, text="Email address:", anchor="w")  self.emailAddrLabel.grid(row=2, column=3)  self.emailAddrEntry = customtkinter.CTkEntry(self.newUserFrame, placeholder\_text="email address...")  self.emailAddrEntry.grid(row=2, column=4, columnspan=2, sticky="nsew", padx=(12), pady=12)  #create a seperator to distuinguish between sections  seperator = customtkinter.CTkFrame(self.newUserFrame, height=1, fg\_color="gray")  seperator.grid(row=3, column=0, columnspan=10, padx=20, pady=20, sticky='nsew')  self.createNewUserButton = customtkinter.CTkButton(self.newUserFrame, text="Create new user", command=lambda:self.logonDB.createUserCreds(self.usernameEntry.get(), self.confirmpasswordEntry.get(), self.accessLevelEntry.get(), self.emailAddrEntry.get()))  self.createNewUserButton.grid(row=4, column=0, sticky="w", padx=(12, 0), pady=12)  self.buttonExit = customtkinter.CTkButton(self.newUserFrame, text="Exit", command=self.onClosing)  self.buttonExit.grid(row=4, column=2, sticky="w", padx=(10, 12), pady=12)  #initialise the logondbhandler for making a new user  self.logonDB = logonDBHandler()  self.logonDB.initializeDatabase()  if (self.confirmpasswordEntry.get() == self.passwordEntry.get()):  self.createNewUserButton.configure(command=lambda:self.logonDB.createUserCreds(self.usernameEntry.get(), self.confirmpasswordEntry.get(), self.accessLevelEntry.get(), self.emailAddrEntry.get()))  else:  self.passwordEntry.configure(text\_color="red")  self.confirmpasswordEntry.configure(text\_color="red")  def onClosing(self, event=0):  self.destroy() | | |

|  |  |  |
| --- | --- | --- |
| pieChart.py – python library source code | Requirement(s) met: **S2, M11, M11A** | Technicality: B |
| *AS MENTIONED PREVIOUSLY IN THIS DOCUMENTATION, “pieChart.py” WAS NOR WRITTEN OR CLAIMED TO HAVE BEEN WRITTEN BY ME. THIS IS CLEARLY DETAILED IN* [*PYTHON LIBRARY/PACKAGE USAGE TABLE OF REFERENCES*](#_Python_library/package_usage) *ON PAGE 15 INCLUDING WHY THE SOURCE CODE NEEDED TO BE MANUALLY DOWNLOADED INSTEAD OF IMPORTING NORMALLY VIA PIP.* | | |
| """  A customtkinter widget for pie chart display.  Author: Akascape  Version: 0.1  Link: https://github.com/Akascape/CTkXYFrame  """  from PIL import Image, ImageDraw, ImageFont  import customtkinter as ctk  import random  import math  class CTkPieChart(ctk.CTkLabel):  def \_\_init\_\_(self,  master,  command=None,  values={},  \*\*kwargs):  self.arc = None  self.im = Image.new('RGBA', (1000, 1000))    self.size = kwargs.get('radius') or 200    self.background = kwargs.get('bg\_color') or master.cget("fg\_color")  self.border\_width = kwargs.get('border\_width') or 0  self.border\_color = kwargs.get('border\_color') or ctk.ThemeManager.theme["CTkButton"]["border\_color"]  self.width = kwargs.get('line\_width') or 20  self.text\_color = kwargs.get('text\_color') or None  self.widget = master  self.values = {}    self.command = command  self.bg = master.cget("fg\_color")    for i in values:  self.add(tag=i, draw=False, \*\*values[i])    super().\_\_init\_\_(master, image=self.arc, fg\_color=self.background, compound='center', text="")  self.draw\_pie\_chart()    def \_set\_scaling(self, \*args, \*\*kwargs):  super().\_set\_scaling(\*args, \*\*kwargs)  self.size = int(self.\_apply\_widget\_scaling(self.size))  self.width = int(self.\_apply\_widget\_scaling(self.width))  self.border\_width = int(self.\_apply\_widget\_scaling(self.border\_width))    def \_set\_appearance\_mode(self, mode\_string):  super().\_set\_appearance\_mode(mode\_string)    def draw\_pie\_chart(self, \*args):    width = self.width \*10  del self.im  self.im = Image.new('RGBA', (1000, 1000))  draw = ImageDraw.Draw(self.im)  draw.arc((0,0, 990, 990), 0, 360, self.widget.\_apply\_appearance\_mode(self.border\_color), self.border\_width)  new\_angle = -90  sum\_ = 0    for i in self.values.values():  sum\_ += i["value"]    for value in self.values.values():  try:  old\_angle = new\_angle  new\_angle = old\_angle + (value['value']/sum\_) \* 360    draw.arc((self.border\_width, self.border\_width, 990-self.border\_width, 990-self.border\_width), old\_angle, new\_angle, value['color'], width)    midpoint\_angle = (old\_angle + new\_angle)/2    xn = yn = (900 - self.border\_width)/2  radians = (990 - self.border\_width)/2  arc\_pos = radians / 3  textpos = arc\_pos/1.5  perc = int(round(value['value']/sum\_ \* 100))    midpoint1\_x = xn + (radians - textpos) \* math.cos(math.radians(midpoint\_angle))  midpoint1\_y = yn + (radians - textpos) \* math.sin(math.radians(midpoint\_angle))    draw.text((midpoint1\_x, midpoint1\_y), text=str(perc)+"%", fill=value['text\_color'],  font=ImageFont.load\_default(size=70))    x0 = width+self.border\_width  x1 = 990-width-self.border\_width    if x0>x1:  x1=x0  draw.arc((x0,x0,x1,x1), 0, 360,  self.widget.\_apply\_appearance\_mode(self.border\_color), self.border\_width)  self.arc = ctk.CTkImage(self.im.resize((self.size, self.size), Image.LANCZOS), size=(self.size, self.size))  super().configure(image=self.arc)  except:  pass  def add(self, tag, value, color=None, text\_color=None, draw=True):    if tag in self.values:  self.update(tag, value, color, text\_color)  return    if color is None:  #color = "#"+''.join([random.choice('ABCDEF0123456789') for i in range(6)])  color = random.choice(['#E7CBA9', '#CCD4BF', '#F5F3E7', '#EEBAB2'])  if text\_color is None:  if self.is\_color\_too\_bright(color):  text\_color = "black"  else:  text\_color = "white"  if self.text\_color:  text\_color = self.text\_color    self.values.update({tag:{'color': color, 'value': value, 'text\_color': text\_color}})    if draw:  self.draw\_pie\_chart()  def delete(self, tag):  if tag in self.values:  del self.values[tag]  self.draw\_pie\_chart()  def update(self, tag, value=None, color=None, text\_color=None):  if tag in self.values:  if value:  self.values[tag]['value'] = value  if color:  self.values[tag]['color'] = color  if text\_color:  self.values[tag]['text\_color'] = text\_color  self.draw\_pie\_chart()  super().update()    def cget(self, param):  if param=="bg\_color":  return self.background  if param=="border\_color":  return self.border\_color  if param=="border\_width":  return self.border\_width  if param=="line\_width":  return self.width  if param=="radius":  return self.size  if param=="width":  return super().winfo\_width()  if param=="height":  return super().winfo\_height()  if param=="values":  return self.values  if param=="text":  raise ValueError(f"No such parameter: {param}")  if param=="justify":  raise ValueError(f"No such parameter: {param}")  if param=="text\_color":  raise ValueError(f"No such parameter: {param}")  if param=="text\_color\_disabled":  raise ValueError(f"No such parameter: {param}")  if param=="corner\_radius":  raise ValueError(f"No such parameter: {param}")  if param=="font":  raise ValueError(f"No such parameter: {param}")  if param=="image":  raise ValueError(f"No such parameter: {param}")    return super().cget(param)  def configure(self, \*\*kwargs):    if "bg\_color" in kwargs:  self.background = kwargs["bg\_color"]  kwargs.update({"fg\_color": self.background})  if "border\_color" in kwargs:  self.border\_color = kwargs.pop("border\_color")  if "border\_width" in kwargs:  self.border\_width = kwargs.pop("border\_width")  if "radius" in kwargs:  self.size = kwargs.pop("radius")  if "values" in kwargs:  self.values = kwargs.pop("values")  for i in values:  self.add(tag=i, draw=False, \*\*values[i])  if "line\_width" in kwargs:  self.width = kwargs.pop("line\_width")    super().configure(\*\*kwargs)  self.draw\_pie\_chart()  def is\_color\_too\_bright(self, hex\_color, threshold=100):  if not hex\_color.startswith("#"): return False    hex\_color = hex\_color.lstrip("#")  r, g, b = tuple(int(hex\_color[i:i+2], 16) for i in (0, 2, 4))  total = (r + g + b) / 3    return True if total > threshold else False  def get(self, tag=None):  if tag:  return self.values[tag]  return self.values | | |

|  |  |  |
| --- | --- | --- |
| popUpWindow.py – popUpWindow UI & functionality | Requirement(s) met: **S1A, S2A** | Technicality: B |
| import customtkinter  import tkinter  from processes.windowSuperClass import \*  class popUpWindow(superWindow):  def \_\_init\_\_(self, message, windowName="Popup Message"):  self.message = message  self.windowName = windowName  def create(self):  self.box = customtkinter.CTk()  self.box.geometry(f"{310}x{50}")  self.box.title(self.windowName)    frame\_new = customtkinter.CTkFrame(master=self.box,width=310, height=50, corner\_radius=10)  frame\_new.place(relx=0.5, rely=0.5, anchor=tkinter.CENTER)  label\_new = customtkinter.CTkLabel(master=frame\_new, width=200, corner\_radius=10, text=self.message)  label\_new.grid(row=0, column=0, sticky="w", padx=(0, 12), pady=12)  confirmButton = customtkinter.CTkButton(master=frame\_new, width=20, corner\_radius=10, text="Dismiss", command=self.onClosing)  confirmButton.grid(row=0, column=1, sticky="w", padx=(0, 12), pady=12)  self.box.mainloop()  def onClosing(self, event=0):  self.box.destroy() | | |

|  |  |  |
| --- | --- | --- |
| sendEmail.py – email sending functionality | Requirement(s) met: **M13, M13A** | Technicality: B |
| #email imports  from email.message import EmailMessage  import smtplib  #general imports  from dotenv import load\_dotenv  import os  class appEmail:  #Load environment variables from the .env file  envVarPath="src/config/.env"  load\_dotenv(dotenv\_path=envVarPath)  #configure private environment variables for the sender account and sender password  \_\_defaultSenderAddr = os.getenv('DEF\_EMAIL\_ADDR')  \_\_defaultSenderAddrPass = os.getenv('DEF\_EMAIL\_ADDR\_PASS')    def \_\_init\_\_(self):  self.mailserver = smtplib.SMTP('smtp.gmail.com',587)  self.mailserver.starttls()  self.mailserver.ehlo()  self.mailserver.login(self.\_\_defaultSenderAddr, self.\_\_defaultSenderAddrPass)  def sendEmail(self, destinationAddr, subject, content):  try:  self.message = EmailMessage()  self.message.set\_content(content)  self.message['subject'] = subject  self.message['to'] = destinationAddr  self.message['from'] = self.\_\_defaultSenderAddr  server = smtplib.SMTP("smtp.gmail.com", 587)  server.starttls()  server.login(self.\_\_defaultSenderAddr, self.\_\_defaultSenderAddrPass)  server.send\_message(self.message)  server.quit()  except Exception as error:  print(f"Error encountered when sending email: {error}")  return False | | |

|  |  |  |
| --- | --- | --- |
| stockLevelChecker.py – stock level checking | Requirement(s) met: **M7, M7A, M7B** | Technicality: B |
| #database imports  from dbHandling.DBHandler import \*  from dbHandling.stockLevelDBHandler import \*  from dbHandling.productDBHandler import \*  from dbHandling.stockLevelHistoryDBHandler import \*  #graphing imports  from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk  from matplotlib.figure import Figure  import numpy as np  import matplotlib.pyplot as plt  #process imports  from processes.sendEmail import \*  class CheckStockCount(DBHandler):  \_\_defaultStockLevelTable = "stocklevel"    def runStockLevelCheckAgainstMinimum(self):  self.cursor.execute(f"SELECT product\_id, stock\_count, minimum\_stock\_level, reOrder\_level FROM {self.\_\_defaultStockLevelTable} WHERE stock\_count <= minimum\_stock\_level")  dataToUse = self.cursor.fetchall()  self.productDBHandler = productDBHandler()  for i in range(len(dataToUse)):  productName = self.productDBHandler.getProductName(dataToUse[i][0])  subject = f"Stock level alert: stock count for '{productName}' is lower than minimum"  content = f"Stock level {dataToUse[i][1]} is lower than minimum stock level {dataToUse[i][2]}. A new delivery is recommended."    emailAlert = appEmail()  emailAlert.sendEmail(self.\_defAlertEmail, subject, content) | | |

|  |  |  |
| --- | --- | --- |
| windowSuperClass.py – window parent class | Requirement(s) met: **S2, S2A** | Technicality: B |
| import customtkinter  customtkinter.set\_appearance\_mode("System") #modes: "System" (standard), "Dark", "Light"  customtkinter.set\_default\_color\_theme("blue") #themes: "blue" (standard), "green", "dark-blue"  class superWindow(customtkinter.CTk):  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.protocol("WM\_DELETE\_WINDOW", self.onClosing)  self.bind("<Shift-q>", self.onClosing)  self.bind("<Command-w>", self.onClosing)  def onClosing(self, event=0):  self.destroy() | | |

*templates-folder source*

|  |  |  |
| --- | --- | --- |
| envVarTemplate.txt – environment variable template | Requirement(s) met: **n/a** | Technicality: n/a |
| DB\_USERNAME=your\_username  DB\_PASSWORD=your\_password  DB\_HOST=your\_host  DB\_SCHEMA=your\_database  DEF\_EMAIL\_ADDR=your\_sender\_email\_addr  DEF\_EMAIL\_ADDR\_PASS=your\_sender\_email\_pass  DEF\_ALERT\_RECIPIENT\_EMAIL=your\_default\_email\_addr\_for\_receiving\_app\_alerts | | |

*weeklyReports-folder source*

|  |  |  |
| --- | --- | --- |
| weeklyReport\_xx-xx-xxx.txt – weekly report output file | Requirement(s) met: **n/a** | Technicality: n/a |
| *n/a, this is an output file for weekly reports* | | |

### Requirements met

|  |  |
| --- | --- |
| *Requirement* | *Page number* |
| **M1**: The application must be able to run a counting process where a staff member goes around the shop and tallies each product that is needed and then records this information (stock\_count) in the product database, updating previous stock counts. | 47, 48, 79, 81 |
| **M1A**: The record stock delivery functionality must have an easy to use and understand UI and enough labels to accurately describe what each input is needed for. | 47, 48 |
| **M1B**: Staff should be able to record a stock count and then the program update the respective stock levels for each product. | 47, 48, 79 |
| **M1C**: Stock levels must be able to be displayed in a legible manner. | 50, 51 |
| **M1D**: The change in stock level for the current week should be displayed/shown when a specific product entry is clicked on. | 50, 51 |
| **M1E**: Any changes to the stock level of a product need to also be stored in a stock level history table, so that the data can be analysed for the weekly report. | 81 |
| **M2:** The application must be able to create products. | 53, 55, 77 |
| **M2A**: Based on the user access level (shown in User privileges based on interview), staff should be able to create and edit details about products (product\_id, supplier\_id, product\_name, product\_description, product\_pack\_size, product\_weight, product\_barcode, product\_price) within the “product” database, allowing for accurate information at all times. | 53, 55, 77 |
| **M2B**: The new product should be added to the database table for products, and any necessary UI elements that rely on the products to be displayed need to be updated in order to show the entry of the new product into the system. | 53, 55, 77 |
| **M3**: The application must be able to create suppliers. | 55, 82 |
| **M3A**: Based on the user access level, staff should be able to create and edit details about suppliers within the “suppliers” database, allowing for accurate information at all times (stock\_id, product\_id, stock\_count, minimum\_stock\_level, reorder\_level, lastDelivery). | 55, 82 |
| **M3B**: The functionality should update any areas where suppliers need to be selected, such as the data view functionality or where there are any lists that use supplier names, so that data is consistent throughout the system without needing to close and re-open the application. | 55, 82 |
| **M4**: The application must be able to record stock deliveries. | 43, 45 |
| **M4A**: The record stock delivery functionality must have an easy to use and understand UI, such as having a date picker to save the user having to input a date manually and enough labels to accurately describe what each input is needed for. | 43 |
| **M4B**: On days when there are stock deliveries, the app must be able to record what has been delivered and in what quantity. This should automatically update stock levels for the correct products. | 45 |
| **M5**: The application must present analytics/state information on sales. | 50, 51 |
| **M5A**: The application must display stock counts and any necessary information about products, suppliers and waste in a legible manner. A useful way to present this would be in a table, with a search function that highlights the search term if it exists, and a button appears which allows you to see the graph detailing the trend in stock level for that product (if the user is on the products tab, for example). | 50 |
| **M6**: The application must be able to record any waste (this includes but is not limited to stolen, perished and damaged items). | 58, 59, 84 |
| **M6A**: There must be a separate tab in the application for recording any waste. This should include a “resolvement” status (for example, green being resolved, red being unresolved and yellow being in progress). | 58, 59 |
| **M6B**: Waste should be able to be “resolved” (the status of the waste record to be updated to true or false). The user should be able to choose a waste product and update its respective status. | 51, 58, 59 |
| **M7**: The application must notify you when re-orders are needed. | 70, 108 |
| **M7A**: Re-orders could be based on a set date or only trigger a notification when a minimum stock level is reached. | 70, 108 |
| **M7B**: A notification of this need for a re-order should be sent by email to the default email address. | 108 |
| **M8**: The application must be able to highlight fast and slow-selling items/trends in stock level. | 59 |
| **M8A**: This information could be highlighted in a weekly report or notification. Allows management to make informed decisions on what product to purchase for the store. | 63 |
| **M9**: The application must be able to generate a weekly report. | 61, 63, 84 |
| **M9A**: The program should predict a stock level for each day of the coming week. This prediction should be based on the data of the previous weeks' analysis. | 63 |
| **M9B**: The program should calculate the revenue, cost of goods sold and then net profit. | 63 |
| **M9C**: The program should calculate the trend in the week's stock level for each product using linear regression analysis. | 63 |
| **M9D**: The user should have the option to automatically email this data to the default email address upon generation of a weekly report. | 63 |
| **M9E**: The user should have the option to generate a .txt output of the weekly report once it has been generated. | 63 |
| **M9F**: The user should be able to view previous weekly reports in the database. | 61 |
| **M10**: The application must have a login system, where the user enters their username/unique staff ID and password, and user accounts should be able to be created | 74, 87, 97, 98, 100 |
| **M10A**: The login system must have the functionality of being able to change their password. This can be done by entering their old password, the new one, and confirming the new one. If the user has forgotten their password, they can reset it using a recovery key that is generated for them upon account generation. | 74, 87, 98 |
| **M10B**: The application should have a user access level-based functionality, so that each user access level corresponds to a certain amount of allowed functionality (see User privileges based on interview). | 39, 42, 74, 98, 100 |
| **M10C**: The user must be able to reset their password if they forget it. | 97 |
| **M10D**: Admin users must be able to create new user accounts once they have navigated to the settings pane. | 100 |
| **M11**: The application should provide a visual breakdown of the database tables so that a user can understand what is happening. | 42, 102 |
| **M11A**: This could include a pie chart with a key detailing all the different database tables present. | 42, 102 |
| **M12**: The application should allow admin users access to a settings pane. | 68 |
| **M12A**: The admin should be able to create new users. | 68, 100 |
| **M12B**: The admin should be able to change any environment variables as detailed in environment Variables. | 68 |
| **M13**: The program must have the ability to send emails (provided it has access to the internet). | 107 |
| **M13A**: The program must be able to send emails when necessary to inform staff/admins of any alerts or actions that have occurred in the application. | 107 |
| **S1**: The application should be easy on the eye. | 70, 89, 93, 106, 109 |
| **S1A**: There shouldn’t be too much information on a page at one time. Information should be presented clearly and effectively. Buttons should be large and colourful, and obvious to what action they perform. | 106 |
| **S2**: The application should be intuitive. | 70, 89, 93, 102, 109 |
| **S2A**: A range of staff will be using the application, some more technologically adept than others. Therefore, the program should be as easy to use as possible. | 70, 89, 93, 106, 109 |
| **S3**: The application should store data on a server local to the business. | 71, 74 |
| **S3A**: A suitable location could be the office that the manager and assistant manager use. | 71, 74 |
| **S4:** The application should notify staff members through either WhatsApp, email or text messages | 63, 79, 107 |
| **S4A:** Notifications should be accessible by the entire staff team. The easiest way to do this would be to automate notifications by sending them to the WhatsApp work group chat which all members of staff are a part of and can access easily, or by sending emails to a default email address or to the user email address once they have logged in. | 63, 79, 107 |
| **S5:** Consider using a tabulated view-based menu system for easy navigation, allowing for all staff to be able to understand how to use the app without much prerequisite technical knowledge. | 53, 55, 50, 42, 43, 68, 58, 47, 61 |
| **S5A:** This ease of usability should also be enhanced by having a vertical navigation bar for the tabulated view with larger buttons, aswell as the built in one provided by the tabulated view in *customtkinter*. | 39, 42 |
| **S6:** The program should allow a user to update any environment variables | 68 |
| **S6A:** The user, if logged in as an admin, should be able to edit/change any environment variables found in [envVarTemplate](bookmark://envVarTemplate) | 68 |
| **S7:** Have autocomplete entries for any search bars to make the program easier to use for staff members | 86 |
| **S7A:** This could be useful for auto filling product names etc in the data view UI tab | 86 |

# Testing

## Functionality testing

|  |  |
| --- | --- |
| Testing video URL (TinyURL shortened) | <https://tinyurl.com/18401-8011-Soma-NEA-Testing> |
| Testing video URL (YouTube un-shortened) | <https://www.youtube.com/watch?v=2ZP7SEVjfG8> |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test number | Requirement number | Reason for test | Test type | Value/action | Expected outcome | Actual outcome | Time stamp |
| 1 | **M10, S4** | To test whether a new account can be created | Normal | Username: admin  Password: 12345  Access level: 1  Email: [sbenf999@outlook.com](mailto:sbenf999@outlook.com) | Can see the created values in the database | See the account record in database table | 1:08 |
| 2 | **M10** | To test whether we can log into the application with our credentials | Normal | Login to the application using the following values/credentials:  Username: admin  Password: 12345 | The program redirects us to the main application, and we see the welcome message. | Login is successful and we are brought to the application | 1:30 |
| 3 | **M10** | To test correctly that incorrect values don’t log us in | Rejected | Username: soma  Password: hello | Username and password entry boxes outlined in red | Incorrect username/password popup | 1:46 |
| 4 | **M10** | To test a misspelt username | Boundary | Username: admiN  Password: 12345 | Username and password outlined in red | Incorrect username/password popup | 2:08 |
| 5 | **M3, M3A, M3B, S5** | To create a supplier | Normal | Enter details for a supplier | Supplier is added to the suppliers table | Supplier is added to suppliers table | 3:00 |
| 6 | **M2, M2A, M2B, S5, S7, S7A, S1, S2** | To create a product for this supplier | Normal | Enter reasonable details for a realistic product, including minimum and reorder levels at 10 and 15 respectively. | Product is added to the database correctly | Product is added to products table | 3:53 |
| 7 | **M4, M4A, M4B, S5, S7, S7A, S1, S2** | Record a delivery for the newly created product | Normal | Enter a value of 20 for the newly created product in the delivery. | Stock count for the product should be increased by 20 and reflected in the stock level history table | Stock count is increased from 0 to 20, with action “delivery” in stock level history table | 4:28 |
| 8 | **M1, M1A, M1B, M1C, M1E, S5, S7, S7A, S1, S2** | Carry out a stock count for the newly created product | Normal | Enter a value of 15 for the newly created product in the stock count | Stock count for the product should be set to 15 | Stock count is set to 15, stock level history table shows this and “count” action | 5:18 |
| 9 | **M6, M6A, S5, S7, S7A, S1, S2** | Create some waste items | Normal | Enter a quantity of 2 waste items for the previously created waste product, do not select that wase has been resolved. | Waste product record should be in waste table, stock level for the product should decrease by 2 | Waste record created correctly, stock count reflects decrease | 6:08 |
| 10 | **M6B, S5, S1, S2** | Test whether waste can be resolved or not | Normal | Navigate to the data view tab, and then to the waste tab. Search for the previously created product and press the “update resolvement” status button. | Waste resolvement status should update to 1 | Waste resolvement status is updated to the value of 1, signifying change in status | 7:06 |
| 11 | **M6** | Test invalid waste entry | Rejected | Leave required field empty for the waste entry | Error message displayed | Error message pops up | 7:45 |
| 12 | **M7, M7A, M7B, S4** | Verify the functionality of stock level vs reorder/minimum stock level notification functionality. | Normal | Navigate to the stock counting tab and create a stock count of 14 for the newly created product. Exit the program and relaunch it, then check the inbox of the default email address. | Email sent to default email address detailing that the stock level if below the reorder level. | Email notification detailing that the stock count for the product is lower than minimum | 9:39 |
| 13 | **M9, M9A, M9B, M9C, M9D, M9E, S4, S5, S1, S2** | Verify the functionality of the generate weekly report tab | Normal | Navigate to the weekly reports tab, both send email and .txt file output selected, and press the generate button. | Email should be sent to default email address detailing the previous weekly report, there should be a local .txt file | Local .txt file generated correctly, and email sent accordingly, both containing the weekly report information | 10:32, 11:04 |
| 14 | **M9, M9F, S5, S1, S2** | Verify that previous weekly reports can be viewed | Normal | Navigate to the weekly reports tab, select the previous weekly reports and view it in the frame below. | Select weekly report button action should display the data about any products that was calculated for the weekly report. | Weekly report is correctly displayed upon button action inside application, verifiable by weekly report record seen in MySQL workbench | 11:41 |
| 15 | **M12B, S5, S1, S2, S6, S6A** | Verify environment variable change | Normal | Modify variable | Change applies successfully | Variable is updated correctly to change | 13:01 |
| 16 | **M1** | Test invalid stock count | Rejected | Enter negative stock count | Error message displayed | Error message pops up | 13:49 |
| 17 | **M4** | Test invalid stock delivery quantity | Rejected | Enter negative quantity | Error message displayed | Error message pops up | 14:27 |
| 18 | **M10D, S1, S2** | Ensure admin can create new users | Normal | Admin adds new user with user access level of level 2 (this gives reduced privileges in functionality accessibility within the program) | New user created and this can be seen in the user’s table. | New user is successfully created, and this can be seen in MySQL workbench application | 15:42 |
| 19 | **M10B, S1, S2** | Ensure user access level-based functionality is working | Normal | Log into application with user account 1 of access level 2 | Functionality available to user should decrease with access level. | Reduced functionality in program as well as greyed out navigation buttons. | 16:11 |
| 20 | **M10C, S1, S2** | Check password reset mechanism | Normal | Change old password to new, different one | Password resets successfully, login should be successful with new password | Password hash is updated successfully in users table for admin user, ability to login | 17:09, 17:19 |
| 21 | **M1D, S5A, S1, S2** | Check working stock level visualisation | Normal | Navigate to data view UI, search for previously created product and select the visualize button | Program displays stock level graph | Stock level for added product is visualized successfully with respect to previous stock level changes observed in the video. | 17:54 |

## Usability testing

To carry out the usability testing part of the documentation with the client, I organised an in-person meeting/interview to test the application and discuss any future improvements that could be made. I did this by first presenting the application running on my computer and allowing the client to look through the program themselves, during which I explained what each part of the program fulfilled functionality wise. After doing this demonstration, the client suggested some improvements to the program which I have detailed below:

|  |  |  |  |
| --- | --- | --- | --- |
| Improvement number | Improvement | Description of suggestion | How would I implement this? |
| **I1** | Allow products stock to be able to change by weight as well as stock count | The client described this in the scenario of batches of brownies they bake. For example, the store may purchase a 12kg back of brownie mix to be used in the baking process. They detailed that in each batch, ~3kg of the brownie mix is used. As figure is managed by hand with no record, the store often finds themselves running out of brownie mix or realising too late that they need to order more. The client suggested being able to therefore also track products by *weight,* such as this brownie mix. This should also notify them when a re-order is needed. | To implement a tracking of weight, in my opinion this could be achievable by adding *types* of products to the product database. These could consist of ingredients (such as the brownie mix), general products that may be sold on the shop floor and other items. Furthermore, it may be possible to adjust the products table to set a new “trackByWeight” column to be **true,** allowing stock counts to modify the weight value of a product record in a stock count process if the program detects that this Boolean value is set to true. Furthermore, notifications simply needed to be adapted to fit this weight tracking. A minimum weight can be set for these items instead of a minimum re-order level, and when this level is reached, a notification suggesting a re-order can be sent to the default user email address. This simply requires the adaption of the already existing notification system which checks stock levels against the minimum on page 108. |
| **I2** | Allow the user to record sales day by day, with ability to record reduced products aswell. | The OneStop store also sells hot bakery items cooked in-store, such as sausage rolls and bacon-cheese turnovers. These items are not managed through the *OneStop* till system. The client suggested being able to record sales of these items day-by-day, as they are generally instead tallied up by hand on a piece of paper at the end of the week, whereby any profit calculations are made. They should also be able to record reduced price items, such that profit calculations can correctly adjust to any money they may have lost by doing so. | In my opinion this could be implemented by programming similar functionality to that of the stock counting tab, but instead of stock counts, the user enters the amount sold of said items, and if they have been reduced in their price (and if so, to what price). An algorithm should then work out the profit loss on any reduced items and add this to a report which is produced day to day of any sales. This would allow the client to be able to manage any bakery sales on a finer level and also giving them the ability to predict on what days to bake more or less of a certain item. |
| **I3** | Ability to add multiple default recipient email addresses in the environment variable file instead of just one | The client explained that when either the manager, or assistant managers may be away, they might not be able to access certain email addresses which they usually check when in the office. They suggested having multiple email addresses for the default email addresses found in the environment variables file, so that multiple people can receive any notifications about stock level warnings and weekly reports at one time. | This can be simply implemented by using commas to separate multiple values for an environment variable. In the beginning, the environment variable for the default recipient email address should be just one. However, by modifying the settings pane, an admin account should be able to choose to add extra email addresses, for example to a maximum limit of 10. With this taken into consideration, the *sendEmail* class found on page 107 should be modified so that it sends an email ***x*** number of times when called (in a for loop), ***x*** being the number of additional email addresses. This also allows the program to respond dynamically to email address changes, as it is not hard coded in the *sendEmail* class but rather relies on data found in the environment variable file. |
| **I4** | Employee accountability | The client suggested that the details of what actions have been made in the application should be logged by the application, allowing any issues to be brought up with the employee if they are faced. It also allows management to ensure that specific tasks are completed on time, and if not, they can see who didn’t complete them and find out why. | This suggestion could be implemented by modifying DbHandling classes, so that any methods which write to a database table (e.g., a task in the application which require this to happen, such as a stock count), could also write information to a ***taskHistory*** table. Each record in this table could store what method the task originated from/what task was being carried out, when it happened (the time at which the record was written), and who by. This would efficiently keep track of all tasks that have been carried out. To further improve this functionality, administrator accounts could have access to an extra tab which presents the data in this ***taskHistory*** table, with a search function that can sort by date and also name, allowing the admin to review previously recorded tasks. |
| **I5** | Time of day & date for changes in stock level to be considered for predicting changes in stock level trends. | The client suggested the improvement of stock levels also being monitored by date as well as ***time***, instead of just by date. This would allow them to more accurately monitor changes in stock level, for example, when a batch of new bakery items have been baked, put out on the shelf, and old ones which are now cold taken off. This, alongside changes in stock levels, could allow them to predict a specific time when people are more and/or less likely to purchase a specific item. | Whilst the stock level history table already tracks both the date and time at which a stock level for a product was changed, either by a delivery, stock count or waste entry, this is not taken into account for when predicting stock levels for the upcoming week. By taking the date into account in the weekly report generator, we should be able to generate a more in depth prediction, such as specific time ranges for a day, when a product is more likely to be sold. |

# Evaluation

|  |  |
| --- | --- |
| Requirement | **M1** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | For the requirement M1, to be able to run a stock counting process where a staff member goes around the shop and tallies each product that is needed and then tallies that information, I was solving the problem of non-OneStop franchise products to be able to be store their stock counts and respective data in a manner that would help management and staff to manage the products in the store in a more efficient way. I solved this by creating an easy-to-use UI, with an autocomplete search for products, with product names being autocompleted and showing suggestions when one starts typing based on products already in the products table. There is a quantity entry with error checking to ensure that no negative numbers are entered, and this is added to a list from which you can delete items if you need to during a stock count, such as due to an incorrect quantity or product. |
| 2.1) What parts do as they should | All functionality works as intended, products can be deleted from the stock count and quantities changed before submission, stock counts and stock level history is updated correctly and accordingly, and widgets are cleared correctly after submission. |
| 2.2) What parts could be improved | See improvement 1 in *Usability testing* |
| 3) Feedback from usability tests | The UI is simple and easy to use, with the ability to change the quantity before submitting a stock count being extremely useful. The need to confirm the stock count is also helpful if there are any mis clicks. |

|  |  |
| --- | --- |
| Requirement | **M2** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | For the requirement M2, the application must be able to create new products. This was solved by allowing the user to select a supplier who delivers the product and then allowing them to fill out the necessary information about the product. Some of the product data is split across two tables, where information about the product itself goes into the products table and information about its stock count, minimum stock level and reorder level go into the stock level table. |
| 2.1) What parts do as they should | Everything works as it should |
| 2.2) What parts could be improved | No improvements need to be made, other than having the ability to choose the type of product being created (for example an ingredient) so that different metrics can be checked instead of the default stock level (such as weight for ingredients in baking). See improvement 2 in *Usability testing* |
| 3) Feedback from usability tests | The UI is simple and easy to use, no additions needed, and the fact that all widgets are cleared after creation of a product makes the transition between creating another one more seamless. |

|  |  |
| --- | --- |
| Requirement | **M3** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | The problem solved was the need to be able to create new suppliers. The program allows the user to input a name for the supplier, a description of who they are and any other additional information and allows the user to add a list of dates on when the supplier may be delivering products. This is in the form of a date picker and allows the user to choose any number of specific dates and delete them if need be. |
| 2.1) What parts do as they should | Suppliers are always created successfully, if no supplier dates are chosen then this raises an error. |
| 2.2) What parts could be improved | n/a |
| 3) Feedback from usability tests | UI is easy to use and navigate, however it would make more sense to not have set dates, but rather a specific date each month when the supplier is known to deliver, e.g., a period of every 2 weeks for some specific items which don’t require delivery so often. |

|  |  |
| --- | --- |
| Requirement | **M4** |
| 1) Design documentation and design outcome | A screenshot of a computer  AI-generated content may be incorrect. |
| 2) What problem did I solve and how did I solve it? | The problem that I need to solve was being able to record deliveries for products. The UI allows the user to select a supplier for the delivery and allows you to select a date with the date picker. Whilst the date picker automatically fills the entry with the current date (assuming that the date when you carry out the delivery record is the correct date), the user IS still able to choose a different date in case someone forgot to record a delivery. There is an autocomplete search function to allow the user to add products and their respective quantity to the delivery list. You are also able to change the quantity of a product or delete it if it is no longer needed from the delivery list. Once a delivery is confirmed by the user, stock counts for each product are updated accordingly and the stock level history table updates with values for these stock numbers and the action of “delivery” so that the weekly report functionality can distinguish between a delivery, stock count and waste item records. |
| 2.1) What parts do as they should | All parts work as they should, however sometimes items from a previous stock count appear in the delivery list. Whilst they can be deleted, this error could sometimes cause an incorrect stock level for a product. |
| 2.2) What parts could be improved | Not much to be improved upon other than correct widget clearing as sometimes the previous date stays in the entry widget. |
| 3) Feedback from usability tests | Easy to use and clean UI, stock updates to the database are fast and this is reflected in the DataViewUI. |

|  |  |
| --- | --- |
| Requirement | **M5** |
| 1) Design documentation and design outcome | A screenshot of a computer  Description automatically generated |
| 2) What problem did I solve and how did I solve it? | The user needed to be able to visualise data without needing to go into an application such as *MySQL Workbench* and viewing the data manually. Only data worth viewing, such as products, stock level, suppliers and waste data is shown in the extra tabulated view. There is a search function, that if finds the autocorrected search query, will highlight the row in the table and display any functionality below it. For products, this is to visualize the stock level, and for waste, is to update the resolvement status if it is not 1. |
| 2.1) What parts do as they should | The search function and swapping between tabs works as expected. |
| 2.2) What parts could be improved | When you click on a field, this should allow the user to edit any fields that can be edited (everything except primary and foreign keys). However, this was too difficult to implement as the table wouldn’t return information such as the row on which was clicked, simply the value, which wasn’t enough to construct any SQL statements from. |
| 3) Feedback from usability tests | The highlighting of records is also a good addition to improve the intuitiveness of the UI. |

|  |  |
| --- | --- |
| Requirement | **M6** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | OneStop has a lot of waste, simply from customers damaging items or employees themselves, perhaps accidentally dropping something during a delivery. The waste UI allows the user to search for a product using an autocomplete search bar, describe what happened with the waste item, enter a quantity and a resolvement status. The resolvement status is used to detail whether the waste has been dealt with or not, 0 signalling not deal with and 1 the opposite. Multiple waste items can be recorded at once and this all goes into the waste table in the database and is also subtracted from the stock level of said product and inputted into the stock level history table. This is then taken into account by the weekly report algorithm when calculating profit for the past week. |
| 2.1) What parts do as they should | Waste is added correctly to respective databases and stock level is adjusted to reduce current stock for said products. |
| 2.2) What parts could be improved | Resolvement status could send an email notification and dataViewUI could update data table once the waste resolvement status is updated to reflect this change instead of having to reload the program to see this update |
| 3) Feedback from usability tests | Clean and easy to use, however would incorporate items that are taken home by employees at the end of the day, which are not necessarily considered waste unless thrown away. |

|  |  |
| --- | --- |
| Requirement | **M7** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | The client needed to know when re-orders for certain products were needed so that the stock in the store at any given moment is always enough to stock the shelves and not have items missing. To ensure this, there is a stock level checker that loops through all products and their respective stock count and minimum/reorder levels. If the program sees that a stock count is below one of these levels, then it sends an email to the default email address, advising of a re-order. This process is carried out using multi-threading, during when the user is logging in, to ensure the process is run as frequently as possible. |
| 2.1) What parts do as they should | Emails are sent to the default email address, correctly informing of a required stock re-order |
| 2.2) What parts could be improved | The program could predict when a stock level is going to reach or dip below the minimum and send an email detailing this so that deliveries can be placed in the correct time for them to arrive. |
| 3) Feedback from usability tests | Email notifications is extremely useful for knowing when orders need to be placed for items. |

|  |  |
| --- | --- |
| Requirement | **M8** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | The client needed to be able to see fast and slow selling items/trends in stock level, and I solved this by performing linear regression analysis on the stock levels to see the trends in where it was expected to go. This is a small part of the weekly report algorithm; however it helps the client understand which products are fast or slow selling, especially by being able to compare this trend across different weekly reports to form an informed understanding of the trend for a specific product. |
| 2.1) What parts do as they should | Trend is calculated for the product and displayed in a weekly report correctly. |
| 2.2) What parts could be improved | The weekly report algorithm could automatically compare previous weekly reports to see if a product’s trend has changed significantly, and from this the user may be able to determine why. (for e.g., seasonal items). |
| 3) Feedback from usability tests | Easy to understand and useful to be able to visualise a weekly report inside the application itself. |

|  |  |
| --- | --- |
| Requirement | **M9** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | The client needed to application to generate a weekly report. A weekly report would detail the stock counts of the previous week for each product that changed in stock count, the trend, the predicted stock count for each day of the following week and calculations of revenue, profit and cost of goods sold. This was solved by creating a weekly report algorithm which would calculate these values, using the stock level of each changed product in a dictionary and any additional information. The trend was calculated using linear regression analysis, and the user can optionally generate a .txt output of the report in order to print it off or have a local copy, and the report can also be emailed to the default email address(es) stored in the environment variables file. |
| 2.1) What parts do as they should | All parts of the weekly report UI and functionality work as they should |
| 2.2) What parts could be improved | The .txt and email outputs of the weekly report don’t contain the revenue, cost of goods sold and net profit calculations, whilst the in-app records do. So this needs to be adjusted so that the optional outputs include all data in the weekly report. |
| 3) Feedback from usability tests | The weekly report is extremely useful in predicting stock levels for upcoming week and calculating profit for the client, however this could be improved by calculating reports daily to have a better breakdown of product stock levels day by day. See improvement 5 in *Usability testing* |

|  |  |
| --- | --- |
| Requirement | **M10** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | The application has a login system so that there is a level of security before being able to access the application. It requires a username and password, and once these are correct, the user is redirected to the homepage of the program. When user accounts are created, a username, hashed password, hashed recovery code and access level are written to the users table in the database. Login verification is done by comparing a hash of the entered password to the one stored in the database, if they are the same, then the user is logged in. If the user wishes to change their password they can do so with the change password button, given that they remember their old password. If they cannot remember it, they can reset their password using the recovery code that is generated when the account is first generated. This is a six-digit code, the first 3 characters and the second 3 digits, separated by a dash. The user access level of the logged in user changes the functionality of the application, reducing some functions as the user access level goes down. |
| 2.1) What parts do as they should | Login functionality works correctly, and the program responds correctly to changes in user access level. |
| 2.2) What parts could be improved | Improvements could be made by notifying the user by email that their account has been logged into to improve security. |
| 3) Feedback from usability tests | UI is easy to use and login functionality works well |

|  |  |
| --- | --- |
| Requirement | **M12** |
| 1) Design documentation and design outcome |  |
| 2) What problem did I solve and how did I solve it? | The admin needs to be able to create new users and edit any environment variables. This is done by having a create user function, which allows the administrator to create any number of new accounts. It also allows the admin to edit any environment variables found in the environment variable file. |
| 2.1) What parts do as they should | New users can be created, and environment variables can be changed correctly. |
| 2.2) What parts could be improved | The settings page could be improved by allowing multiple email addresses to be stored for the default recipient email address. Users should alsobe able to be deleted. |
| 3) Feedback from usability tests | See improvement 3 in *Usability testing* |

|  |  |
| --- | --- |
| Requirement | **M13** |
| 1) Design documentation and design outcome | A screenshot of a phone  Description automatically generated |
| 2) What problem did I solve and how did I solve it? | The client needed to know when stock levels for products are too low, which means sending emails to detail these notifications to the default recipient email address. |
| 2.1) What parts do as they should | Emails are sent correctly |
| 2.2) What parts could be improved | This notification system could be further improved by having WhatsApp messages sent to the work group chat, as well as emails, so that there is some redundancy in notifications to the employees at OneStop. |
| 3) Feedback from usability tests | See improvement 3 in *Usability testing* |

|  |  |
| --- | --- |
| Requirement | **C1** |
| 1) What is the problem at hand? | The application should create barcodes. Barcodes could be created and assigned to products in the product database. Furthermore, they could be sent to a label printer for products to be able to be labelled. |
| 2) How could this be implemented? | The product database already has a column for a product barcode. Barcodes are typically stored in a string format, being the string of the digits on the barcode. An external python library may be needed to create the barcodes, and this should occur upon the creation of a product. |

|  |  |
| --- | --- |
| Requirement | **C2** |
| 1) What is the problem at hand? | The application could scan product barcodes and show detailed information related to the item upon scanning. |
| 2) How could this be implemented? | There should be a new tab in the tabulated menu system with the ability to scan barcodes. A barcode scanner would need to be connected to the device on which the application is being run, and when a barcode is scanned, it needs to be compared against barcodes in the product table in the database. If a product is found, then information about the product should be displayed. This could also be implemented by allowing stock counts to be based off scanning a barcode, automatically detecting which product it is and asking the user to input a quantity. |

|  |  |
| --- | --- |
| Requirement | **W1** |
| 1) What is the problem at hand? | The application won’t be able to place deliveries. This must be done by the manager or associate manager; the app will have no integration or ability to place orders automatically. This is a manual task, outside of the project scope. |
| 2) How could this be implemented in the future? | Whilst this can’t necessarily be done, the program informs managers of stock levels to when reOrders should be done. If the supplier had an online ordering form, the program may be able to fill out a form using ***BeautifulSoup*** when it detects that a stock level is below the minimum. |

*\*Requirement* ***W2*** *is not taken into consideration as this application cannot be integrated with official, proprietary OneStop software*

## Final thoughts

### Success of my project

In my opinion, the program has been a success for the client and has met all requirements found in the MoSCoW analysis. The client finds the program intuitive and smooth to use and is happy to implement it in the foreseeable future once the improvements in the usability testing have been implemented.

In some areas the program could have been improved, such as having the ability to edit and delete products and suppliers alongside being able to create them. This would allow the user to be able to control the flow of data more accurately, with an emphasis on being able to keep information more up to date.

Overall, I am satisfied with the outcome of my project and look forward to supporting my client with any queries of questions that they may face in its future usage.

# References

Akascape. (2024, January 14). *Akascape*. Retrieved from Github: https://github.com/Akascape/CTkTable

Akascape. (2024, April 25). *Akascape CTkXYFrame*. Retrieved from Github: https://github.com/Akascape/CTkXYFrame

Akascape. (2024, May 10). *CTkPieChart*. Retrieved from GitHub: https://github.com/Akascape/CTkPieChart/tree/main

GameLoop. (2023). *Subventory app*. Retrieved from GameLoop: https://www.gameloop.com/game/business/com.zippyyum.inventoryapp

Geeks for Geeks. (2023, Novemeber 2). *Disadvantages of Python*. Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/disadvantages-of-python/

Google. (2024, 11 7). *Download Android Studio*. Retrieved from Android Studio: https://developer.android.com/studio

Hari Lee. (2019, August 24). *Creating a Barcode Scanner using Android Studio*. Retrieved from Medium: https://medium.com/analytics-vidhya/creating-a-barcode-scanner-using-android-studio-71cff11800a2

Horacio. (2018, May). *Inventoria software reviews*. Retrieved from Software advice: https://www.softwareadvice.com/inventory-management/inventoria-profile/

InvenTree. (2024, October 2). *Inventree*. Retrieved from Inventree: https://inventree.org/

Jet Brains. (2024, September 25). *The Six Most Popular Cross-Platform App Development Frameworks*. Retrieved from Jet Brains: https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-frameworks.html#popular-cross-platform-app-development-frameworks

Matplotlib. (2024, February 13). *Matplotlib - Visualization with python*. Retrieved from Matplotlib: https://matplotlib.org

Max Verwiebe. (2024, December 19). *CTkDatePicker*. Retrieved from Github: https://github.com/maxverwiebe/CTkDatePicker/tree/main

NCH Software. (2008, July 17). *Manage stock with Inventoria Inventory Management Software*. Retrieved from NCH Software: https://www.nchsoftware.com/inventory/index.html#:~:text=Easily%20organize%20and%20track%20the%20inventory%20for%20your%20business%20for

NCH Software. (2024, October 21). *Manage stock with Inventoria Inventory Management Software*. Retrieved from NCH Software home: https://www.nchsoftware.com/inventory/index.html

Numpy. (2024, December 8). *NumPy*. Retrieved from NumPy: https://numpy.org

OneStop Ltd. (2019, October 19). *OneStop Stores - About us*. Retrieved from OneStop: https://www.onestop.co.uk/about-us/#:~:text=At%20One%20Stop,%20our%20mission%20is%20to%20provide%20convenience%20and

O'REILLY. (n.d.). *Pros and Cons of Tkinter*. Retrieved from O'Reilly: https://www.oreilly.com/library/view/python-programming-on/1565926218/ch20s01s02.html

Philipp Lackner. (2022, August 3). *Local Notifications in Android - The Full Guide (Android Studio Tutorial)*. Retrieved from YouTube: https://www.bing.com/videos/riverview/relatedvideo?&q=how+to+create+an+app+notification+in+java+and+android+sudio&&mid=80514D4351AC14C2CF3F80514D4351AC14C2CF3F&&FORM=VRDGAR

Programmingempire. (2024, April 25). *Features and Benefits of Tkinter*. Retrieved from Programming Empire: https://www.programmingempire.com/features-and-benefits-of-tkinter/

PYPI. (2024, October 15). *mysql-connector-python*. Retrieved from PYPI: https://pypi.org/project/mysql-connector-python/

PYPI. (2024, October 2024). *pip*. Retrieved from PYPI: https://pypi.org/project/pip/

PYPI. (2024, January 23). *python-dotenv*. Retrieved from PYPI: https://pypi.org/project/python-dotenv/

Python - Open Source. (2025, January 8). *hashlib - Secure hashes and message digests*. Retrieved from Python docs: https://docs.python.org/3/library/hashlib.html

Python - Open source. (2025, January 16). *json — JSON encoder and decoder*. Retrieved from Python: https://docs.python.org/3/library/json.html

Python - Open source. (2025, January 16). *smtplib — SMTP protocol client*. Retrieved from Python: https://docs.python.org/3/library/smtplib.html#module-smtplib

Python - Open source. (2025, January 16). *time — Time access and conversions*. Retrieved from Python: https://docs.python.org/3/library/time.html#module-time

ReallyNotAVeryClever. (2023, November 6). *Some thoughts, suggestions and critique after half year use in a small company*. Retrieved from Reddit: https://www.reddit.com/r/InvenTree/comments/17p8bak/some\_thoughts\_suggestions\_and\_critique\_after\_half/

TheDoodleDud. (2024, September 3). *Subventory*. Retrieved from Google Play: https://play.google.com/store/apps/details?id=com.zippyyum.inventoryapp&hl=en\_GB

Tom Schimansky. (2024, September 15). *CustomTkinter*. Retrieved from Github: https://github.com/TomSchimansky/CustomTkinter

Yang bocheng, D. P. (2023, February 14). *Advantages and disadvantages of MAUI*. Retrieved from Github: https://github.com/dotnet/maui/discussions/13310

ZippyZum. (2024, October 17). *Google Play*. Retrieved from Subventory - Apps on Google Play: https://play.google.com/store/apps/details?id=com.zippyyum.inventoryapp&hl=en-US

ZippyZum. (n.d.). *Subventory*. Retrieved from ZippyZum: https://www.subventory.com/Home/Index

1. *WhatsApp* is a free messaging and video calling platform used by OneStop as a means of communication to staff using a *group-chat*. [↑](#footnote-ref-2)
2. *PIP is the package installer for Python -* (PYPI, 2024) [↑](#footnote-ref-3)
3. ***CTkPieChart*** *does not have a pip-installable package, and therefore the source-code is pasted into the source folder for the program. This is detailed clearly within the program and the technical implementation itself.* [↑](#footnote-ref-4)
4. ***CTkXYFrame*** *does not have a pip-installable package, and therefore the source-code is pasted into the source folder for the program. This is detailed clearly within the program and the technical implementation itself.* [↑](#footnote-ref-5)
5. ***CTkDatePicker*** *does not have a pip-installable package, and therefore the source-code is pasted into the source folder for the program. This is detailed clearly within the program and the technical implementation itself.* [↑](#footnote-ref-6)