DATA ANALYTICS PROJECT REPORT

DEPARTMENT OF COMPUTING & INFORMATION TECHNOLOGY – 2024 INTERNSHIP PROGRAM

Student ID: 816030089

Name: Christophe Gittens

Company: First Citizens Bank

STUDENT DETAILS

Christophe Gittens	
Student ID	816030089
Name	Christophe Gittens
Company	First Citizens Bank
Department	Group Internal Audit
Supervisor	Julia Maria Daniel
For Credits (Y/N)	Y

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

A number of people have contributed towards making this internship experience an indelible one. I would like to thank all of my colleagues for making this experience truly memorable and helping us deliver incredibly useful internal tools, which would allow First Citizens' Group Internal Audit to achieve its organizational objectives more efficiently. A special thanks to:

Julia Maria Daniel (Manager – Internal Audit Technology Security and Change Management)

Hannah Taylor (Audit Manager Ag. – Banking & Support Services)

Maurisa Reyes (Audit Assistant)

Clifrine Evans (Manager – Corporate Investigations)

Tracy Bailey (Senior Investigator)

Kathy-Ann Scantlebury (Investigator/Intelligence Officer)

And much thanks to the rest of the Group Internal Audit Team, Banking & Support Team and Investigations Team.

TABLE OF FIGURES

INTRODUCTION

Over this 10 week internship myself and a fellow intern, Ethan Lee Chong, were able to deliver four (4) applications for three (3) separate projects. This document outlays each project's problem, requirements, design and implementation. The projects undertaken during this period are outlined in the table below.

Project	Our Solution
i. Continuous Auditing Project	A Python application which allows the user to upload data stored in text files and excel files and takes user input for a threshold then subsequently obtains the staff transactions subject to review
ii. User Access Verification Project	A Python application which allows the user to upload data stored in text files and takes user input to search for specific user IDs, Customer Information File numbers (CIFs) and/or account numbers and maps the system accesses in one file to transactions stored in another
iii. File Formatting Project	A Python application which allows the user to upload data stored in text files and format them so that the data for each entry is on a single line and not spilling over onto new lines

All of the projects undertaken during the period have undergone user acceptance testing and have been implemented for use within the organization. The Continuous Auditing application has been used to complete the Continuous Auditing activity for the fourth financial quarter of 2024.

PROBLEM

The continuous auditing activity is an organizational function done to monitor the Bank's internal clients (staff) and it is guided by the Central Bank of Trinidad and Tobago's (CBTT's) compliance framework for Anti-Money Laundering and Combatting of Terrorism Financing. Its function is to carry out thorough due diligence of staff related transactions in keeping with the stipulated regulatory requirements as well as internal policies.

Our goal was to provide further automations for the existing continuous auditing procedure to simplify and streamline the process. The existing procedure consisted of a script written in Audit Command Language (ACL) and an Excel sheet with macros written to import, combine, format and export the ACL data to another Excel file. Achieving our goal entailed:

i. Combining these segmented processes into one solution and make a user friendly application with a Graphical User Interface.
ii. Providing a performance boost with our solution as the ACL script was slow and inefficient.
iii. Reducing the data output as the ACL script would include data not subject to the review process.

Functional requirements
i. The user should be able to enter a date for the review period
ii. The user should be able to enter a transaction threshold
iii. The user should be able to upload the input files to the application
iv. The user should be able to select the directory to send the results of the application to
v. The user should be able to append the data of the current review period to a master sheet which contains data for all previous review periods
vi. The user should be able to rerun the application after each session is complete
vii. The user should be given proper feedback for error handling
viii. The system should filter out unwanted transactions e.g. dividend transactions, ACH transactions from insurance companies, transactions from First Citizens Trustee Services and transactions from the Central Bank of Trinidad & Tobago
ix. The system should aggregate transactions based on the Customer Information File number (CIF number)
x. The system should perform aggregations for debit transactions and credit transactions separately

Non-Functional requirements
i. The system should be intuitive with a GUI to interact with the underlying program
ii. The system should provide a performance enhancement of at least 20%
iii. The system should handle errors gracefully
iv. The user interface should be responsive to user input and provide immediate feedback
v. The system should store all data only temporarily and locally
vi. The system should perform all operations on the stored data locally on disk
vii. The system should maintain the integrity of the data

DESIGN

AN OBJECT ORIENTED APPROACH

For our Python solution we opted for an object oriented design which consists of classes, class attributes and methods. Our application contains eleven (11) separate classes:

OpenFiles	FilterAggregateData
StaffAccounts	CombineFormatData
StaffTransactions	ExportFiles
StaffLoans	Main
StaffDeposits	AppGUI
StaffOutsideInterest	

SPRINT 1

The first six (6) classes as well as the Main class were built in sprint cycle one (1) and they were reverse-engineered from the ACL script and mimic its functionality.

SPRINT 2

The FilterAggregateData class was built in sprint cycle two (2) and performs the additional filtering on the data produced from the classes before and also aggregates the debit transactions and credit transactions within the data set.

SPRINT 3

The remaining classes (CombineFormatData, ExportFiles, AppGUI) were built in sprint cycle three (3). They allow for the functionality to combine the data from the FilterAggregateData output and format it with proper column titles to export. It also contains the method to append the formatted data to the master sheet. The ExportFiles class provides the functionality to export delimited text files and excel files. The AppGUI provides the user interface to interact with the entire application. It consists of buttons labels, text fields and date time calendar.

OVERVIEW OF CLASSES

Class	Method Type: decorator/parameter | count	Description
OpenFiles	Overridden Constructor (Y/N): N Class Methods: @classmethod: 4 Static Methods: @staicmethod: 1 Getter Methods: (): 6	This class reads the input files and stores the data in a pandas dataframe, it skips over some header information which may exist in the file
StaffAccounts	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 4	This extracts the accounts related to staff from the data
StaffTransactions	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 4	This extracts the transactions which involve staff from the data
StaffLoans	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 14	This class filters for specific loan codes from the data and find transactions which entail transfers between staff accounts
StaffDeposits	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 7	This class filters for specific deposit codes from the data and find the sum of each based on CIF
StaffOutsideInterest	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 5	This class is used to find the staff with a declared outside interest i.e. some business which falls outside the purview of “The Bank”
FilterAggregateData	Overridden Constructor (Y/N): N Class Methods: @classmethod: 0 **FCTT** Static Methods: @staicmethod: 7 **FCBB** Static Methods: @staicmethod: 1 Instance Methods: (self): 0	This class provides additional filters on the data as well as performs the aggregations [for the Barbados application only the aggregation method is present]
CombineFormatData	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0	This class formats the filtered and aggregated data before export. It

	Static Methods: @staicmethod: 1 Instance Methods: (self): 2	also contains the static method to append the formatted data to the master sheet.
ExportFiles	Overridden Constructor (Y/N): N Class Methods: @classmethod: 0 Static Methods: @staicmethod: 5 Instance Methods: (self): 0	This class has methods to export the data as delimited text files and excel files
Main	Overridden Constructor (Y/N): N Class Methods: @classmethod: 0 Static Methods: @staicmethod: 1 Instance Methods: (self): 0	This class contains one method which calls an instance of CombineFormatData
AppGUI	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 10 Static Methods: @staicmethod: 4 Instance Methods: (self): 0	This class is where the entire application is run from, it provides a graphical user interface for the user to interact with

THE GRAPHICAL USER INTERFACE (GUI)

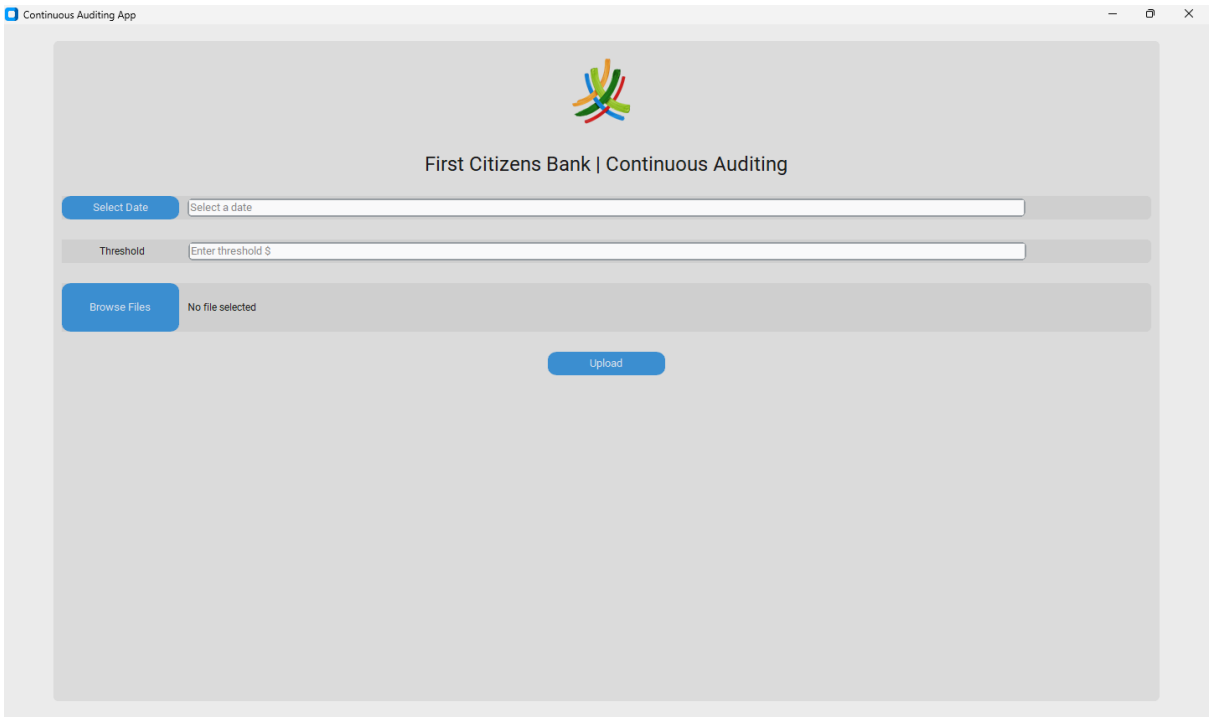


Figure 1: The Launch Page

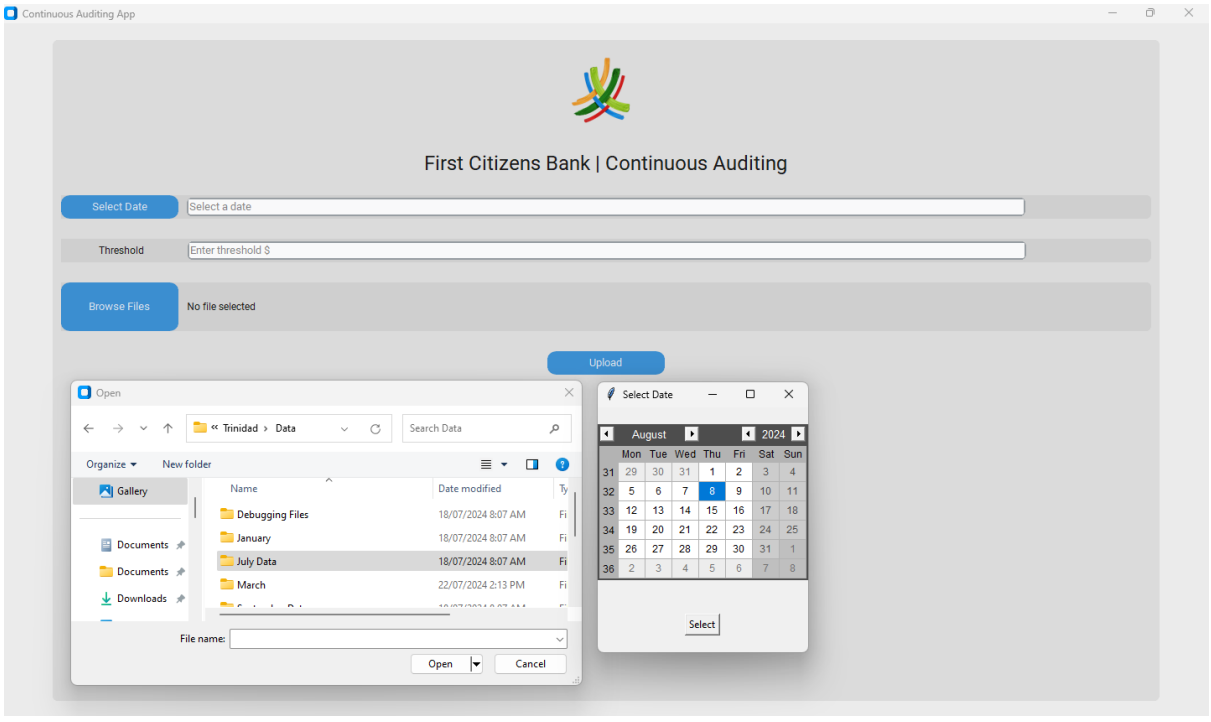


Figure 2: Browse Files Button clicked | Select date button Clicked

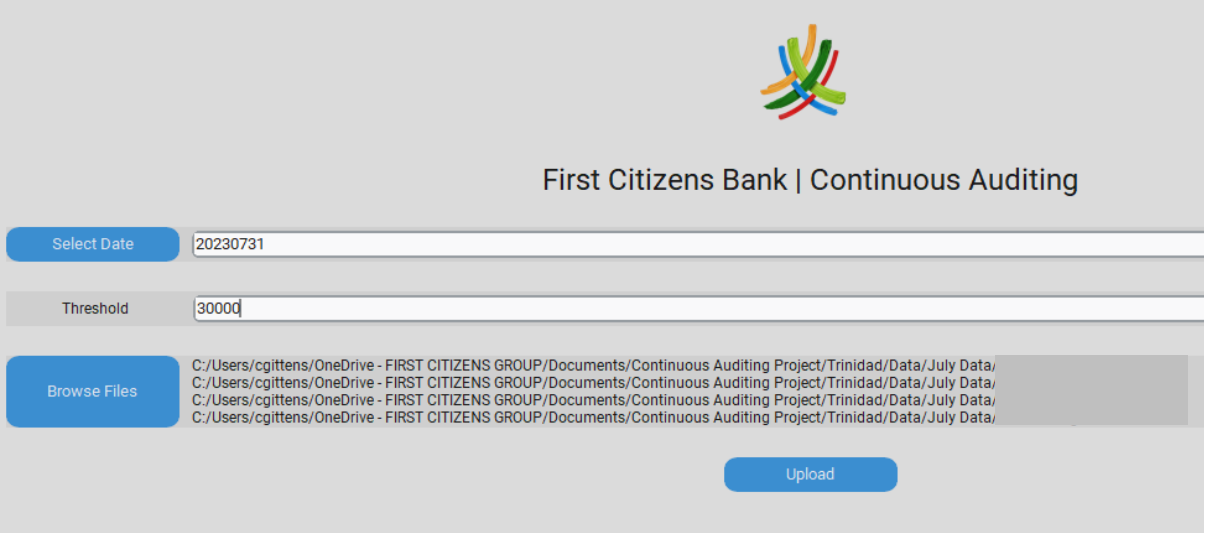


Figure 3: Files chosen | Data in text fields entered

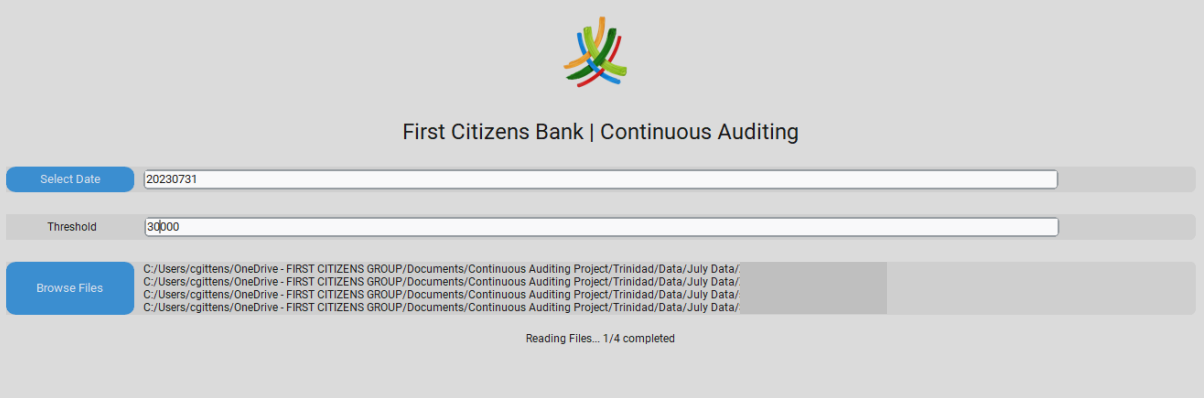


Figure 4: Upload button clicked | Progress label feedback

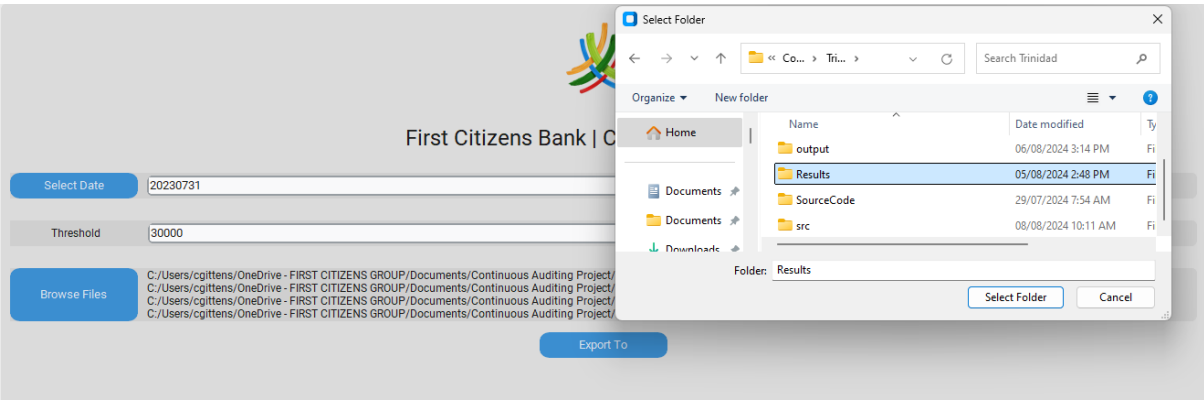


Figure 5: Export To button clicked | File explorer for user to choose destination directory for output

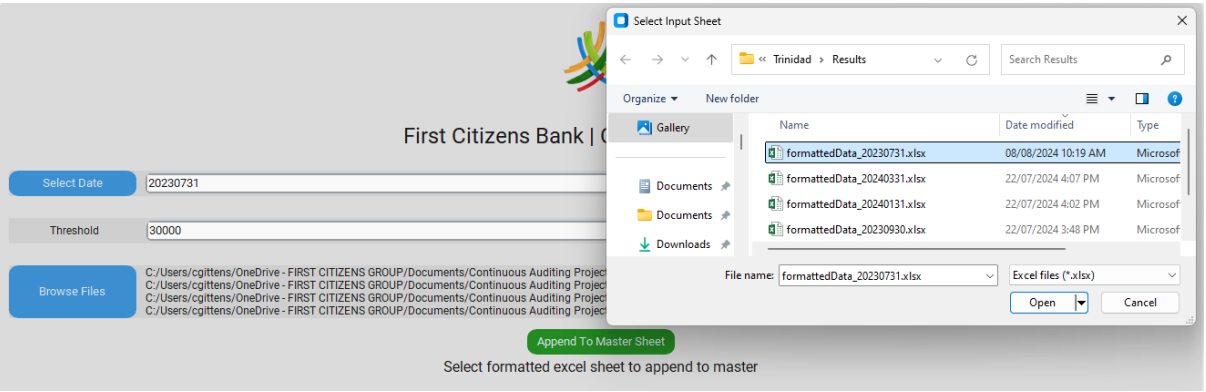


Figure 6: Append to Master Sheet button clicked | User prompted to first select the formatted data sheet

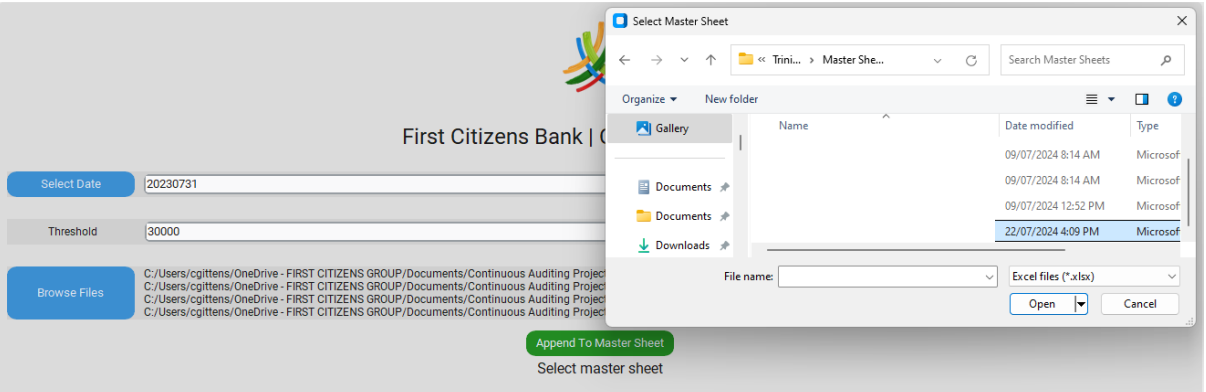


Figure 7: User then prompted to select the master sheet

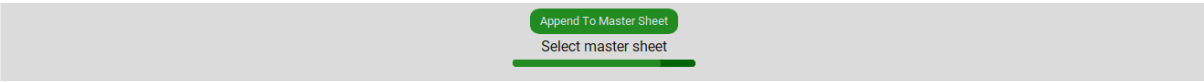


Figure 8: Progress bar feedback when appending

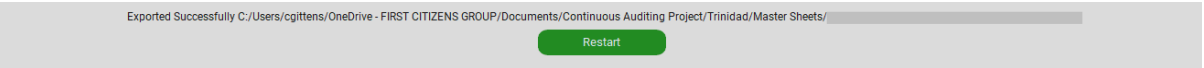


Figure 9: Successful export with feedback | Restart button to begin a new session

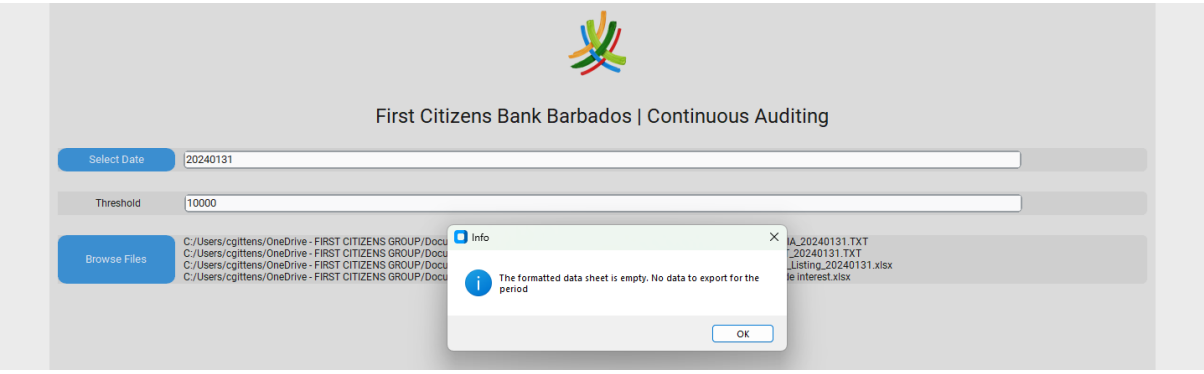


Figure 10: User feedback when there exists no data subject to review

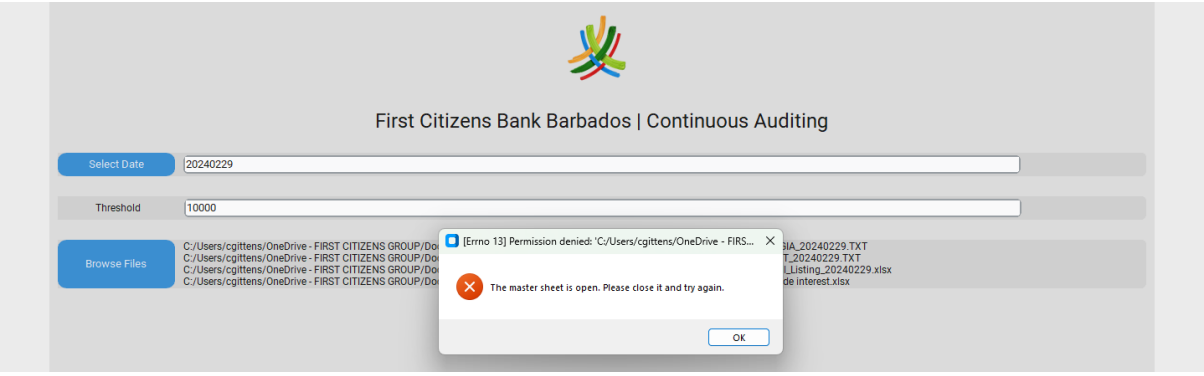


Figure 11: Error Handling | application does not have write permissions to write to the master sheet because it is open

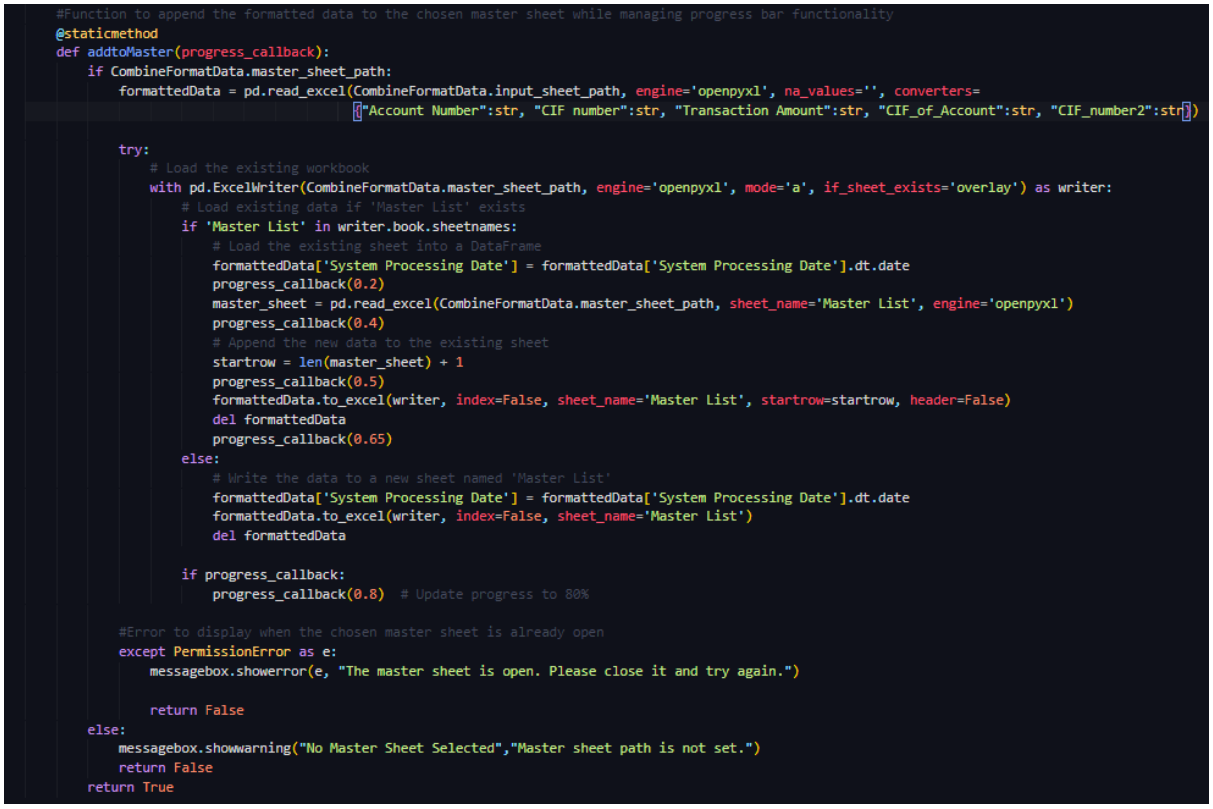
THE PROCESS

Upon launch the user is greeted with a screen with several buttons and labels where they are permitted to enter the review period, enter a threshold value and browse the files needed for the review, the upload button then interacts with the OpenFiles class to read the files and instantiate the class's date and threshold attributes. The user is provided feedback on a label in the GUI of the upload's progress.

Then, once the file upload and attribute instantiation is complete, an Export To button appears on the GUI. This button calls a runner method in Main within which calls a method in the CombineFormatData class using an instance object of CombineFormatData. Each of these methods make subsequent calls to other methods using instance objects to execute the operations on the data. The classes involved are as follows:

StaffAccounts	StaffOutsideInterest
StaffTransactions	FilterAggregateData
StaffLoans	CombineFormatData
StaffDeposits	ExportFiles

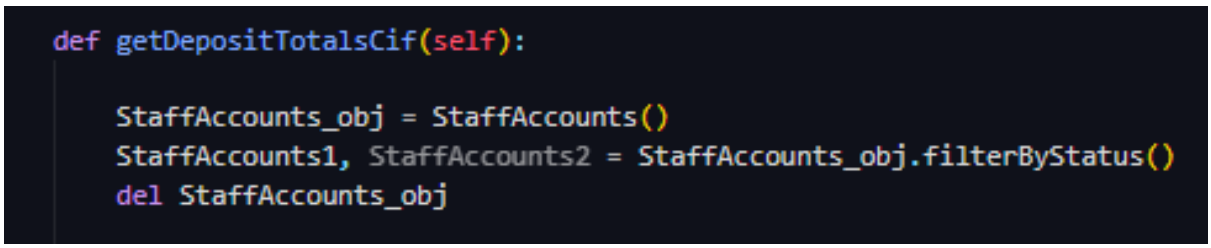
Once that series of procedures is complete the user is greeted with an Append to Master Sheet button. This allows the user to first select the formatted sheet produced by the program and then select the master sheet to append the data to. This functionality is provided by invoking the services of the addToMaster() method in the CombineFormatData class.



```python
#Function to append the formatted data to the chosen master sheet while managing progress bar functionality
@staticmethod
def addtoMaster(progress_callback):
    if CombineFormatData.master_sheet_path:
        formattedData = pd.read_excel(CombineFormatData.input_sheet_path, engine='openpyxl', na_values='', converters=
                                      {"Account Number":str, "CIF number":str, "Transaction Amount":str, "CIF_of_Account":str, "CIF_number2":str})

        try:
            # Load the existing workbook
            with pd.ExcelWriter(CombineFormatData.master_sheet_path, engine='openpyxl', mode='a', if_sheet_exists='overlay') as writer:
                # Load existing data if 'Master List' exists
                if 'Master List' in writer.book.sheetnames:
                    # Load the existing sheet into a DataFrame
                    formattedData['System Processing Date'] = formattedData['System Processing Date'].dt.date
                    progress_callback(0.2)
                    master_sheet = pd.read_excel(CombineFormatData.master_sheet_path, sheet_name='Master List', engine='openpyxl')
                    progress_callback(0.4)
                    # Append the new data to the existing sheet
                    startrow = len(master_sheet) + 1
                    progress_callback(0.5)
                    formattedData.to_excel(writer, index=False, sheet_name='Master List', startrow=startrow, header=False)
                    del formattedData
                    progress_callback(0.65)
                else:
                    # Write the data to a new sheet named 'Master List'
                    formattedData['System Processing Date'] = formattedData['System Processing Date'].dt.date
                    formattedData.to_excel(writer, index=False, sheet_name='Master List')
                    del formattedData

                if progress_callback:
                    progress_callback(0.8)  # Update progress to 80%

        #Error to display when the chosen master sheet is already open
        except PermissionError as e:
            messagebox.showerror(e, "The master sheet is open. Please close it and try again.")

            return False
    else:
        messagebox.showwarning("No Master Sheet Selected","Master sheet path is not set.")
        return False
    return True
```

Image 1: addToMaster method

At the end of the program the restart button resets any class attributes in the OpenFiles class and AppGUI class, during execution of the program objects are deleted right after usage to save space.



```python
def getDepositTotalsCif(self):

    StaffAccounts_obj = StaffAccounts()
    StaffAccounts1, StaffAccounts2 = StaffAccounts_obj.filterByStatus()
    del StaffAccounts_obj
```

Image 2: example of object deletion after it's state is returned

USER ACCESS VERIFICATION PROJECT

PROBLEM

The user access verification activity is a process which aims to determine the legitimacy of system access by staff. To determine legitimacy, accesses may be mapped to a corresponding transaction or be related to a change on a customer's file e.g. a change of address. The goal is to find accesses which do not map to a transaction, from that listing the investigations team can determine and ascertain the legitimacy of those system accesses by performing follow up reviews.

Our goal was to build a system which would assist in automating this mapping. The current process involves a manual search of either the user ID, CIF number or account number within the files. The task is both tedious as well as error prone. Our solution entailed automatically merging the transactions and accesses based on three common fields (The user ID, the transaction date and the account number) which exist in two of the input files. If an access on CIF was done we would first need to obtain all account(s) related to that CIF which is located a separate file and perform the mapping on the account number(s) in that extracted list. Achieving our goal entailed:

i. Providing a Graphical User Interface to interact with the application
ii. Reading the files correctly, fixing some formatting and storing the data in a dataframe
iii. Creating different classes for each type of search
iv. Displaying the data to the user within the application
v. Implementing the functionality to perform concurrent searches and continually append the data of each search
vi. Implementing the functionality to export the data to an excel file containing accesses with no transactions mapped and those with mapped transactions

Functional requirements
i. The user should be able to search for a specific user ID or multiple user IDs
ii. The user should be able to search for a specific CIF or multiple CIFs
iii. The user should be able to search for a specific Account number or multiple Account numbers
iv. The user should be able to perform a search on both CIF and user ID and/or multiple for each
v. The user should be able to perform a search on both Account number and user ID and/or multiple for each
vi. The user should be able to define a date range to review
vii. The user should be able to upload the files to perform the review
viii. The user should be given proper feedback for error handling
ix. The user should be able to view the accesses without a corresponding match
x. The user should be able to reset the application and reupload new data to search
xi. The system should perform a mapping of transactions from one file and system accesses in a another

Non-Functional requirements
i. The system should be responsive to user input and provide immediate feedback
ii. The system should be intuitive to navigate
iii. The system should handle errors gracefully
iv. The system should store all data only temporarily and locally
v. The system should perform all operations on the stored data locally on disk
vi. The system should maintain the integrity of the data

DESIGN

AN OBJECT ORIENTED APPROACH

Our solution to the problem is a Python application with an object oriented design composed of classes, methods and class attributes. Our application contains Nine (9) separate classes:

ReadFiles	SearchByAccNo	Main
SearchByUserID	SearchByUserIDAccNo	AppendData
SearchByCIF	SearchByCIFUserID	AppGUI

SPRINT 1

The ReadFiles class was built in sprint cycle one (1). It reads and stores the inputted files in a pandas dataframe. It also fixes the formatting of the file prior to storing it.

SPRINT 2

The following six (6) classes were built in sprint cycle two (2). Each class is its own search type, when a type of search is specified it calls the relevant method in Main which calls the relevant search class using an instance object of that search class.

SPRINT 3

The remaining classes (AppendData and AppGUI) were built in sprint cycle three (3). They allow for the functionality to perform concurrent searches and appending the results of each search to previous searches. It also contains the methods to append the results to an excel file. The accesses with a corresponding transaction are appended to one sheet in an Excel workbook entitled Matches and those without are appended to a sheet entitled Exceptions. The AppGUI provides the user interface to interact with the entire application. It consists of buttons labels, text fields and two date time calendars to set the date range.

OVERVIEW OF CLASSES

Class	Method Type: decorator/parameter | count	Description
ReadFiles	Overridden Constructor (Y/N): N Class Methods: @classmethod: 3 Static Methods: @staicmethod: 1 Getter Methods: (): 8 Setter Methods: (): 5	This class reads the input files and stores the data in a pandas dataframe, it skips over some header information which may exist in the file
SearchByUserID	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 3	This class performs the user ID search by first filtering the data for the date range then filtering for the user IDs and performing a merge operation on the fields with the same user ID, account number and date
SearchByCIF	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 4	This class performs the CIF search by first obtaining all account numbers related to that CIF (or those CIFs), filtering the data for the date range, filtering the data based on the list of account numbers and then performing a merge operation on the fields with the same user ID, account number and date
SearchByAccNo	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 3	This class performs the account number search by first filtering the data for the date range then filtering for the account number(s) and performing a merge operation on the fields with the same user ID, account number and date
SearchByUserIDAccNo	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 3	This class performs the account number AND user ID search by first filtering the data for the date range then filtering for both the account number(s) AND user ID(s) and performing a merge operation on the fields with the same user ID, account number and date

SearchByCIFUserID	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 3	This class performs the CIF number AND user ID search by first obtaining all accounts related to that CIF (or those CIFs), filtering the data for the date range then filtering the data based on the list of account number(s) AND user ID(s) and performing a merge operation on the fields with the same user ID, account number and date
Main	Overridden Constructor (Y/N): N Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 7	This class runner class is where the respective searches are called from, it also contains method which calls the AppendData class
AppendData	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 5 Instance Methods: (self): 0	This class contains the methods to append the data to an Excel file and the methods to append the data of concurrent searches to a dataframe for the duration of a given search session
AppGUI	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 21	This class is where the entire application is run from, it provides a graphical user interface for the user to interact with

THE GRAPHICAL USER INTERFACE (GUI)

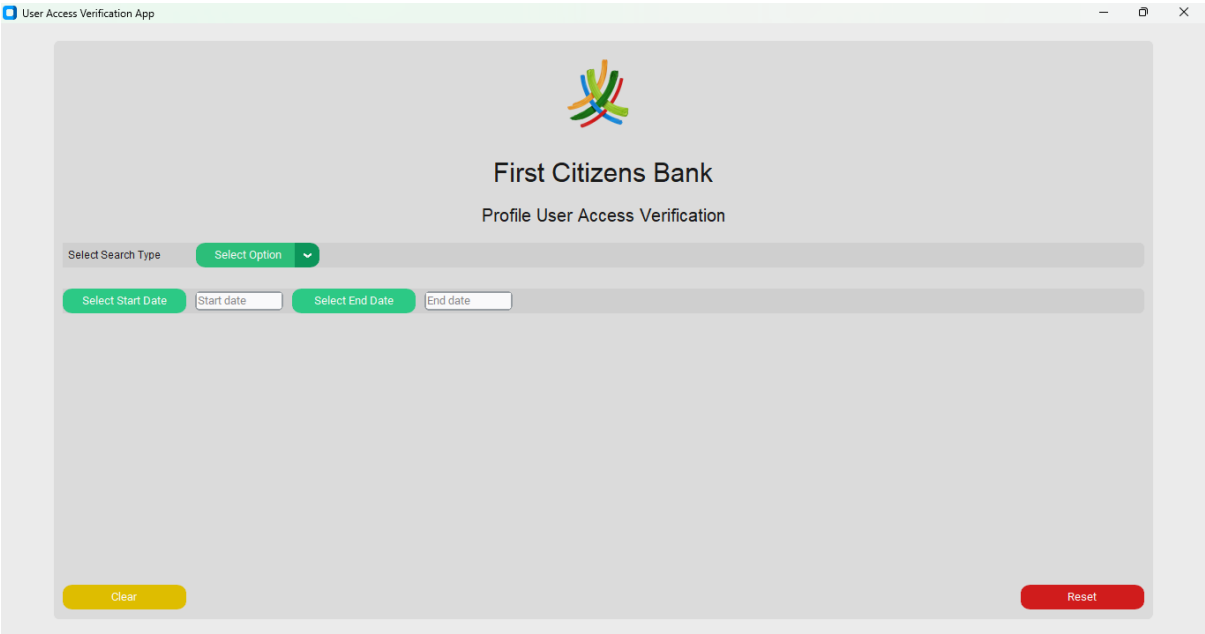


Figure 12: The launch page

Figure 13: Dropdown Menu

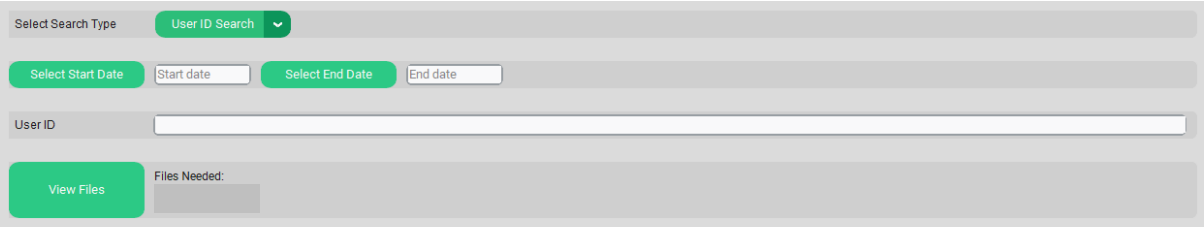


Figure 14: User ID search and user feedback for needed input files

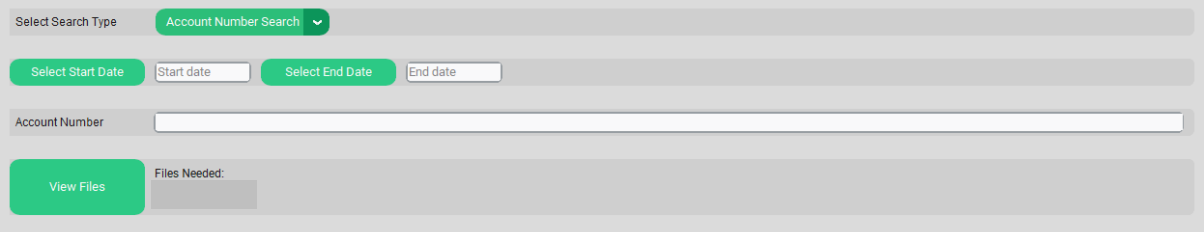


Figure 15: Account Number search and user feedback for needed input files

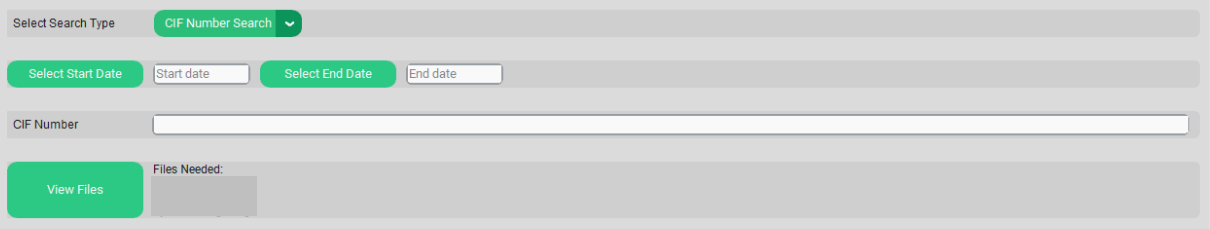


Figure 16:CIF Number search and user feedback for needed input files

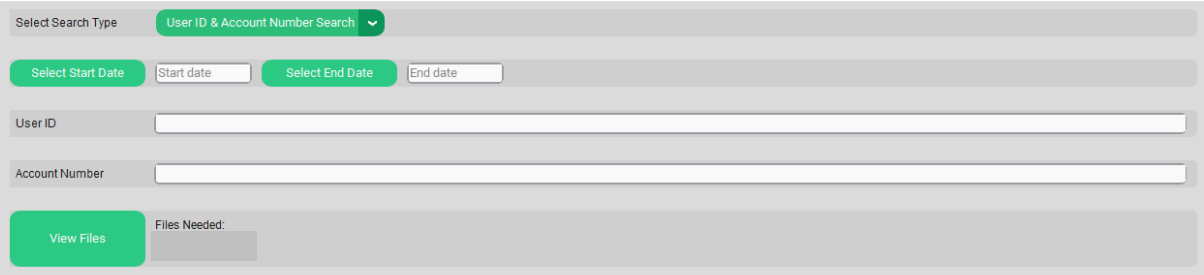


Figure 17: User ID & Account Number search and user feedback for needed input files

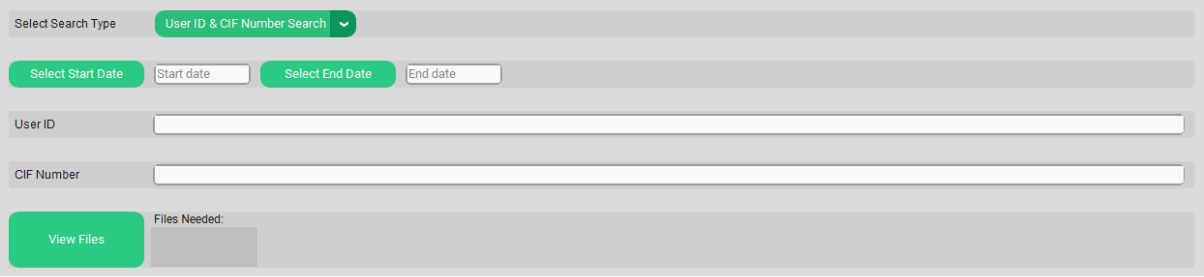


Figure 18: User ID & CIF Number search and user feedback for needed input files

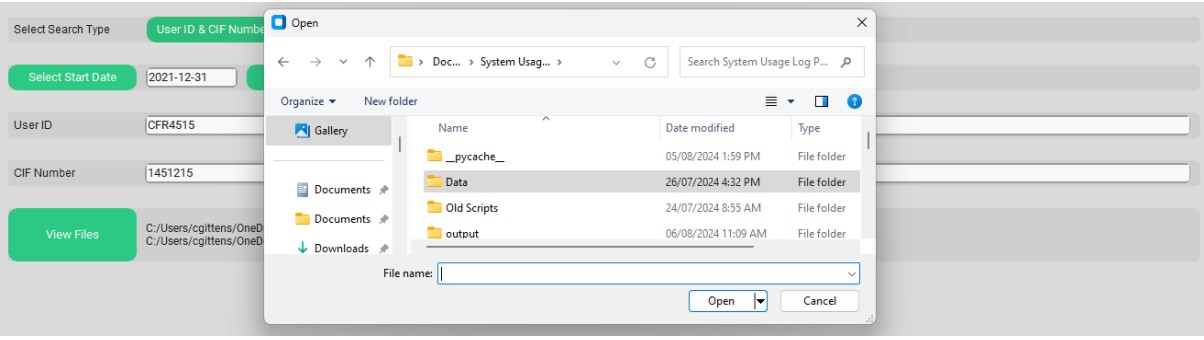


Figure 19: View Files button clicked

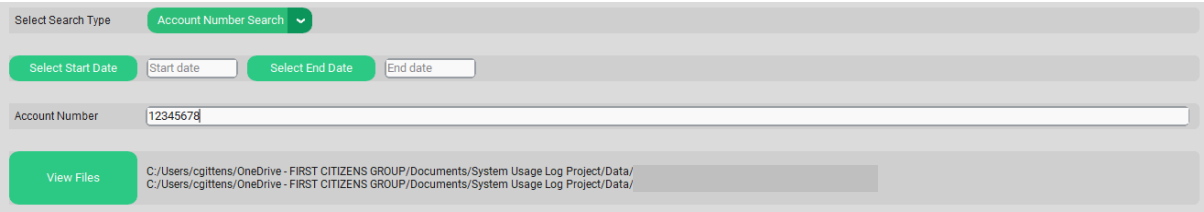


Figure 20: Search for one account number

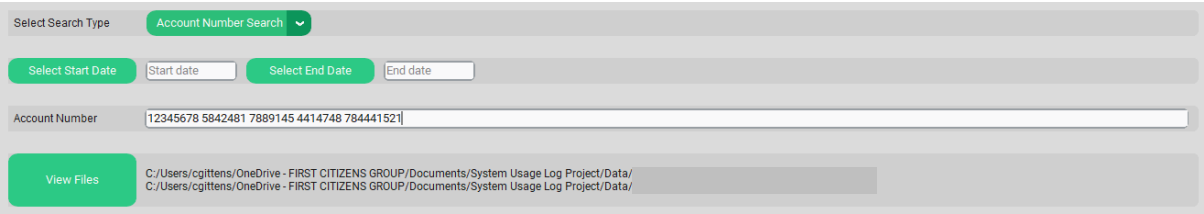


Figure 21: Search for multiple account numbers

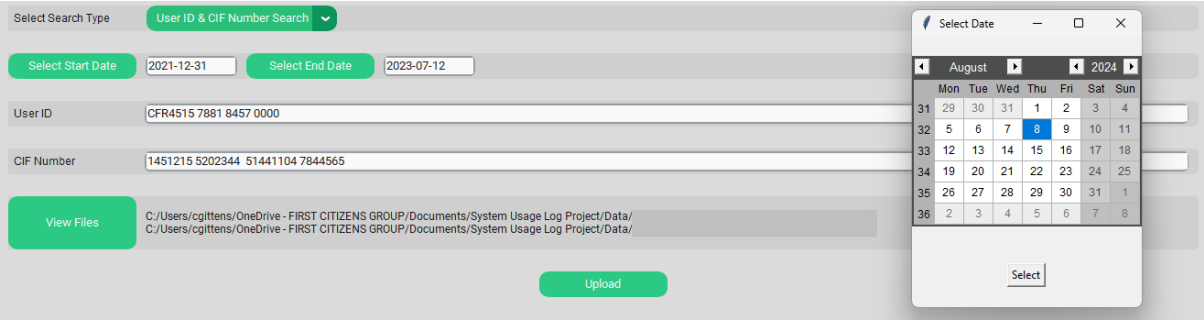


Figure 22: Search multiple values in two fields

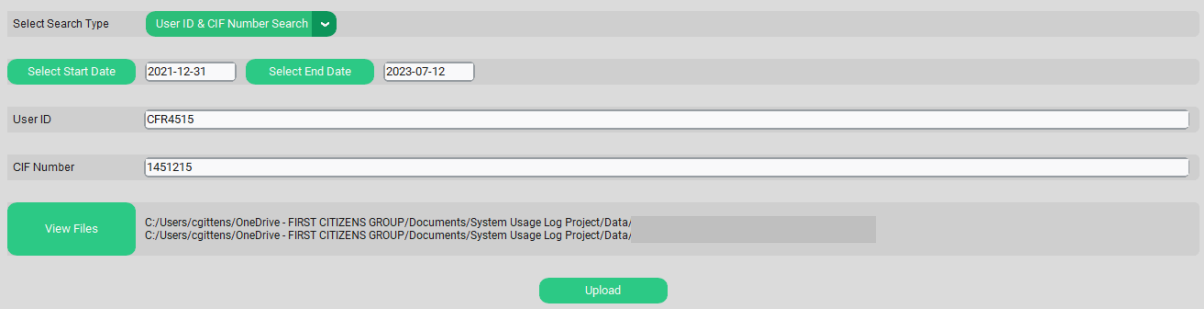


Figure 23: Search single values in multiple fields

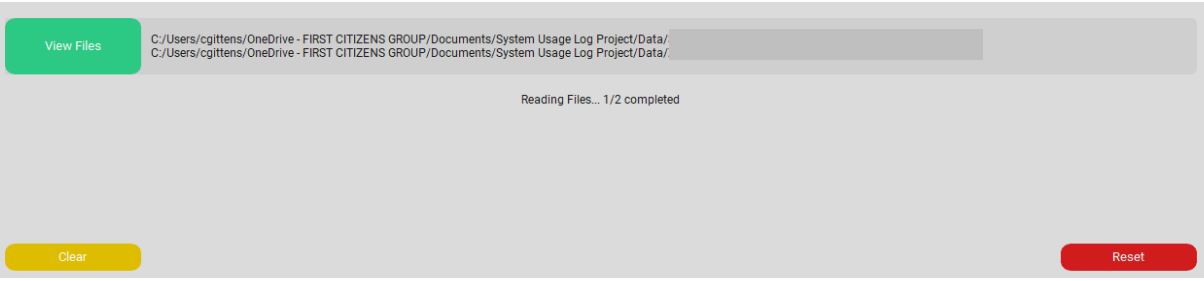


Figure 24: Upload feedback to user

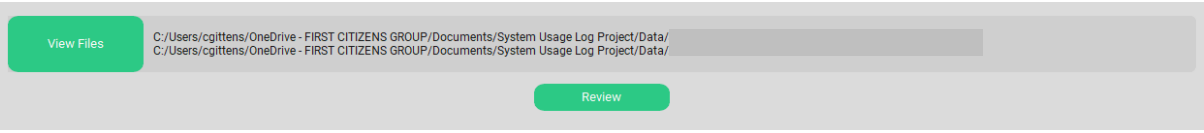


Figure 25: Review button | when clicked performs mapping

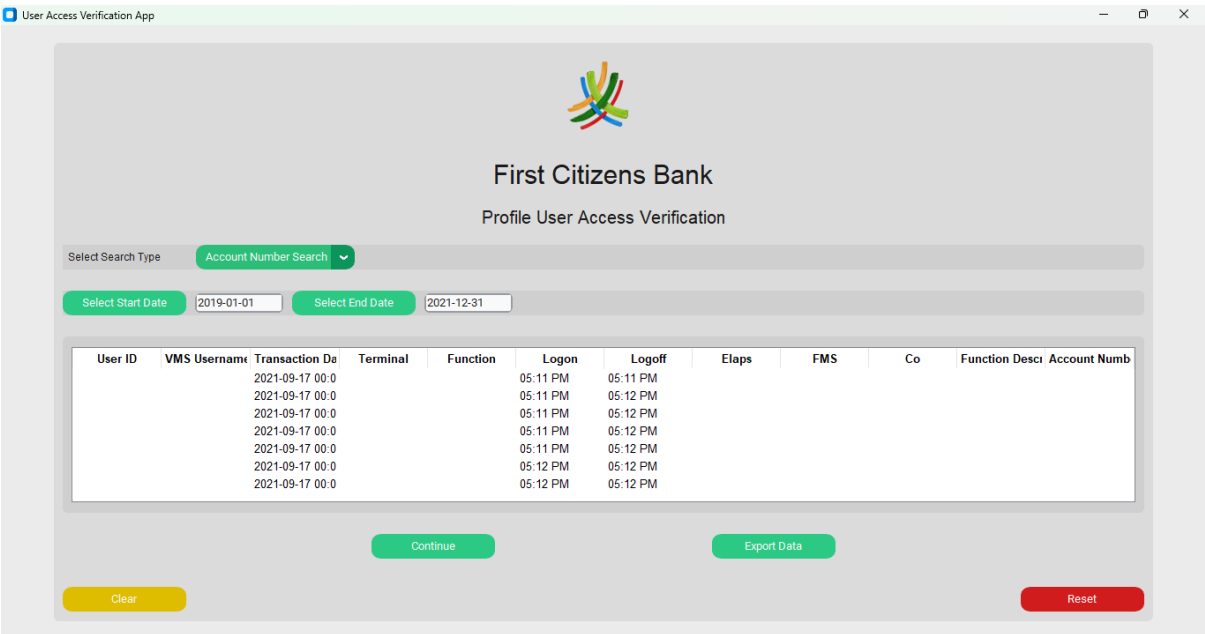


Figure 26: Treemap display within label, displaying accesses without matches | Continue button to continue with the data and an Export Data button to select an excel file to export the data to

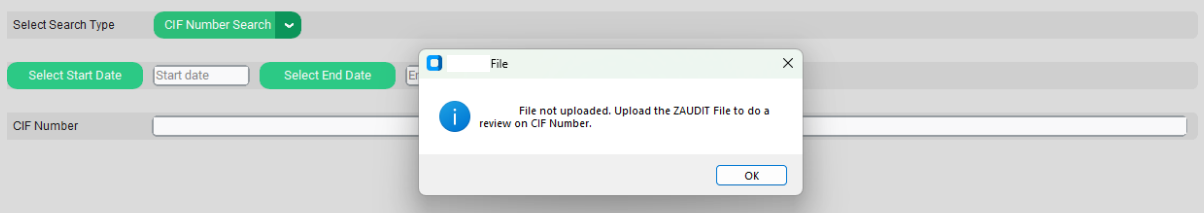


Figure 27: When doing a search which involves using the CIF number, if the file containing that number was not used in the prior search the user is prompted to upload it

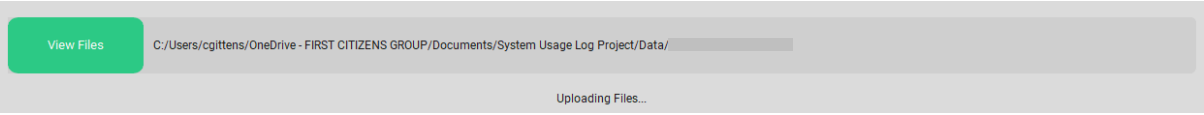


Figure 28: File label when uploading the single missing file which contains the CIF

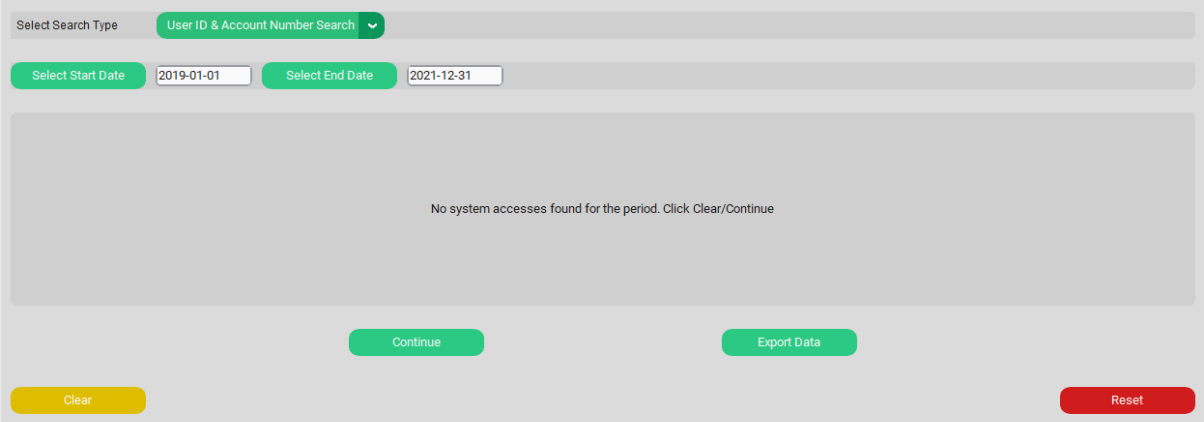


Figure 29: If no accesses were found for the inputted data

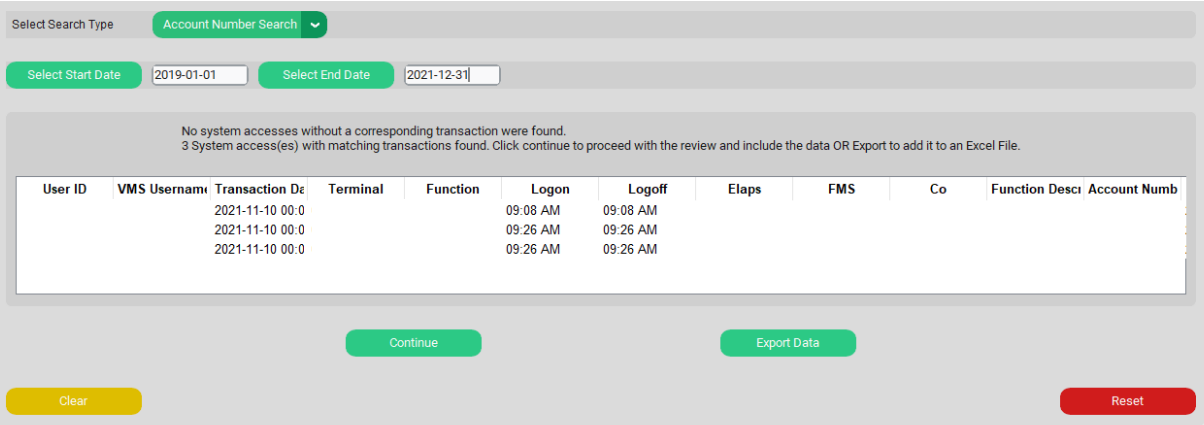


Figure 30: If only accesses with matching transactions is found that information is relayed to the user and displayed in the Treemap

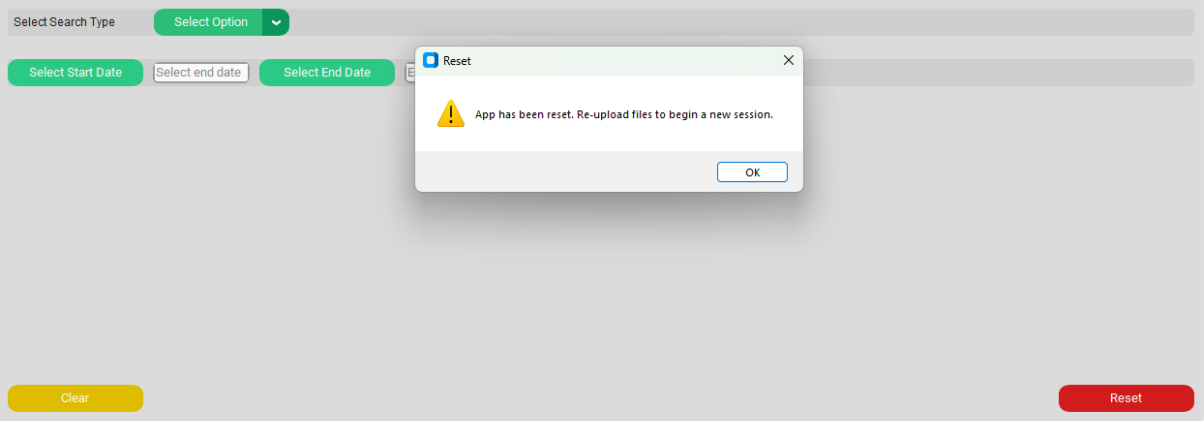


Figure 31: Reset button clicked | the user can upload new data

THE PROCESS

The user selects from the dropdown menu the type of search they wish to perform, based on that selection the relevant labels, text fields and buttons will be displayed to the user. The user can enter a start and end date for the review and the relevant search keys; user ID(s), Account number(s), CIF number(s) or a combination of each depending on the type of search. The user can enter multiple values or just one in any of the fields. Under the hood the program converts the user input to a list using methods native to python.

```
if(AppGUI.choice == "User ID Search"):
    ReadFiles.set_UserID(self.userIDField.get().strip().split())
```

Image 3: Snapshot off GUI class when the Review button is clicked

- The .get() method obtains the value in the respective field, in this instance user ID.
- The .strip() method removes leading and trailing whitespaces
- The .split() method performs the string to list conversion making the user input iterable.

The user would then click the View Files button to launch the file explorer, observing the files that are needed listed on the file label. The user can then upload the files and receive ongoing feedback during file upload. The user can then click review which would display the labels found in either Figure 26, Figure 29 or Figure 30.

Figure 26 – would show in the instance where there are accesses without corresponding transactions

Figure 29 – would show if there are no accesses found for the inputted data and search keys

Figure 30 – would show if all accesses had a corresponding transaction

The user can then select:

- The Continue button to proceed with the search results and do another search.
- The Export Data button to export the current search data as well as all past search data from the current session to an Excel file.
- The Clear button if they wish not to proceed with the current search data.

If the user wishes to get rid of the current data and upload new data to perform searches on they can click the Reset button.

FILE FORMATTING PROJECT

PROBLEM

Data stored in a text file format can sometimes be difficult to analyse due to data spillover in new lines, recurrent header information and overall poor formatting. This makes performing any meaningful analytics on this data particularly tedious

The goal of this project was to clean up data in a .txt file format, make it more readable and store it in a .xlsx format. Achieving our goal entailed:

i. Creating a Graphical User interface to seamlessly interact with the application
ii. Correctly reading the inputted files
iii. Reformatting the file's instances of data trailing on new lines
iv. Omitting header information

Functional requirements
i. The user should be able to select the file type for upload
ii. The user should be able to upload the respective file
iii. The user should be able to preview the contents of the formatted file before export
iv. The user should be able to select a file directory to send the formatted Excel file to
v. The user should be able to upload and format more than one file in the same session

Non-Functional requirements
i. The system should be responsive to user input and provide immediate feedback
ii. The system should be intuitive to navigate
iii. The system should handle errors gracefully
iv. The system should store all data only temporarily and locally
v. The system should perform all operations on the stored data locally on disk
vi. The system should maintain the integrity of the data

DESIGN

AN OBJECT ORIENTED APPROACH

Our solution to the problem was a Python program with an Object Oriented design, consisting of classes, methods and class attributes. Our application contains five (5) separate classes:

FormatAHR
FromatUsageLog
ExportFile
Main
AppGUI

SPRINT 1

The first two (2) classes as well as the Main class were built in sprint cycle one (1) and they and they allow the program to read and store the file in a dataframe

SPRINT 2

The AppGUI class was built as well as the ExportFile class in sprint cycle two (2). It allows for the functionality to format files, view a preview of the formatted file's content, export the file and perform concurrent formatting by reselecting the desired file type from the dropdown menu. Users can upload the file by either using the View Files button or dragging and dropping the file into the label from an separate file explorer.

OVERVIEW OF CLASSES

Class	Method Type: decorator/parameter | count	Description
FormatAHR	Overridden Constructor (Y/N): N Class Methods: @classmethod: 1 Static Methods: @staicmethod: 2 Instance Methods: (self): 0	This class reads and formats the data within it, putting all data that was spilled over onto a single row and column
FormatUsageLog	Overridden Constructor (Y/N): N Class Methods: @classmethod: 1 Static Methods: @staicmethod: 0 Instance Methods: (self): 0	This class reads and formats the data within it, putting all data that was spilled over onto a single row and column
ExportFile	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 1	This class has methods to export the data as an excel file
Main	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 2	This class is where the relevant formatting method is called for the file type
AppGUI	Overridden Constructor (Y/N): Y Class Methods: @classmethod: 0 Static Methods: @staicmethod: 0 Instance Methods: (self): 13	This class is where the entire application is run from, it provides a graphical user interface for the user to interact with

THE GRAPHICAL USER INTERFACE (GUI)

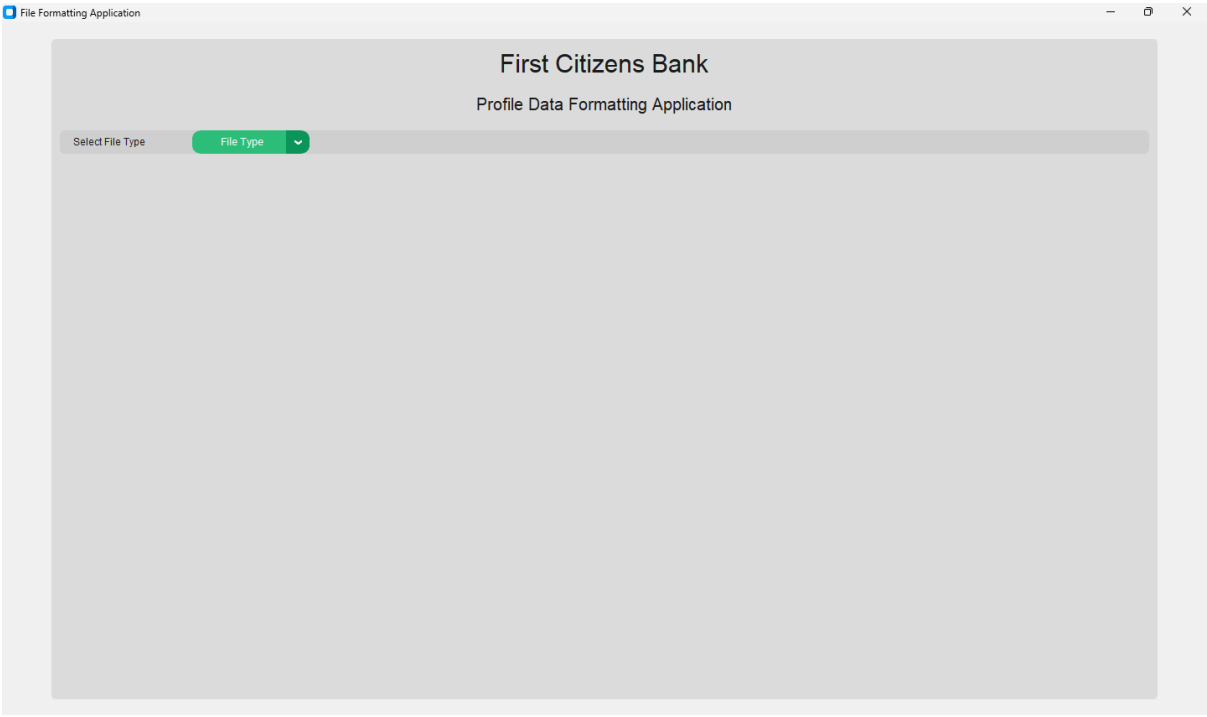


Figure 32: The Launch page

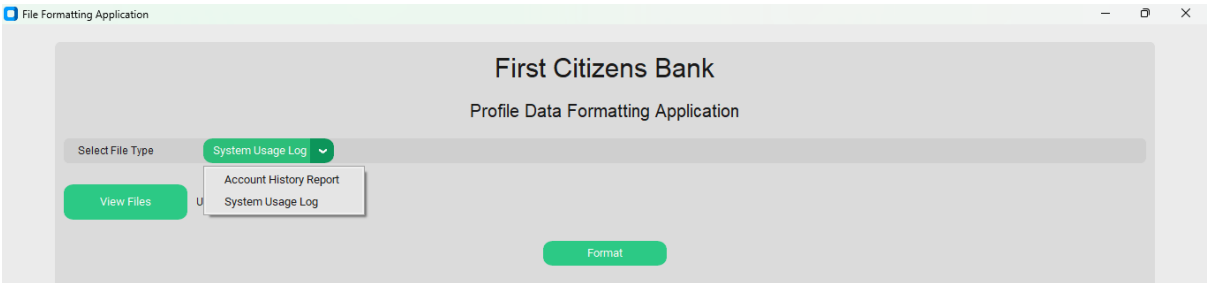


Figure 33: Dropdown menu

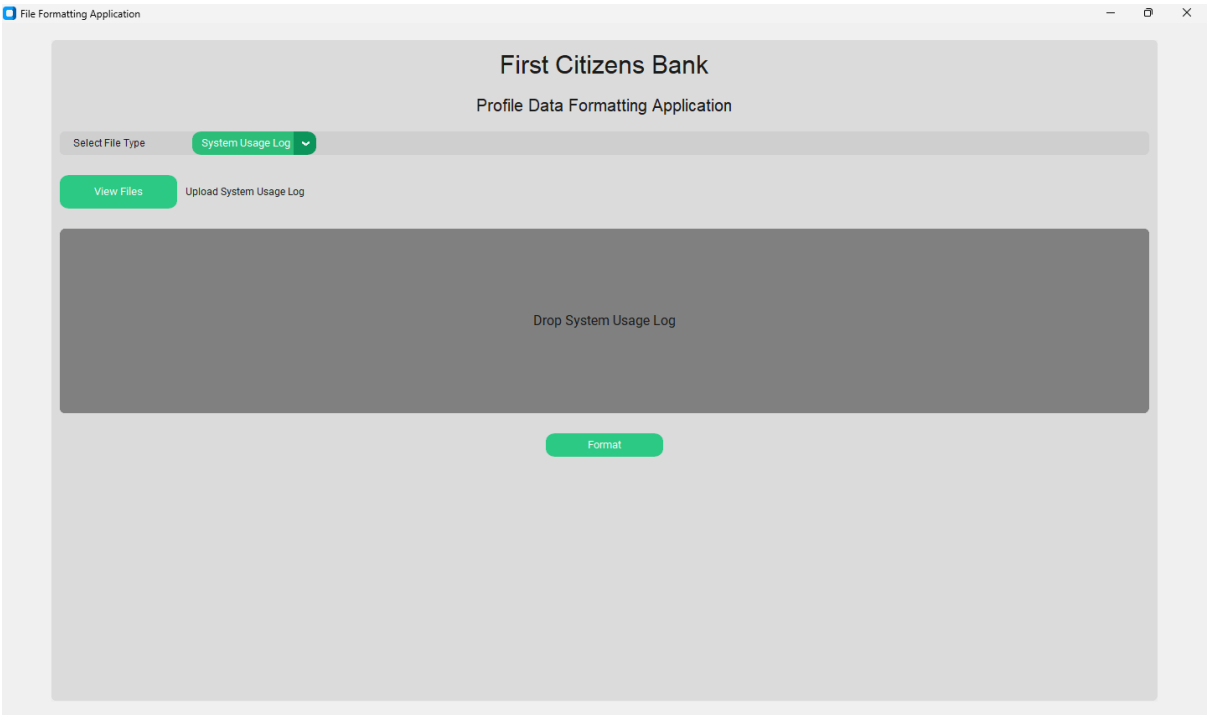


Figure 34: File type selected | User prompted to upload the files

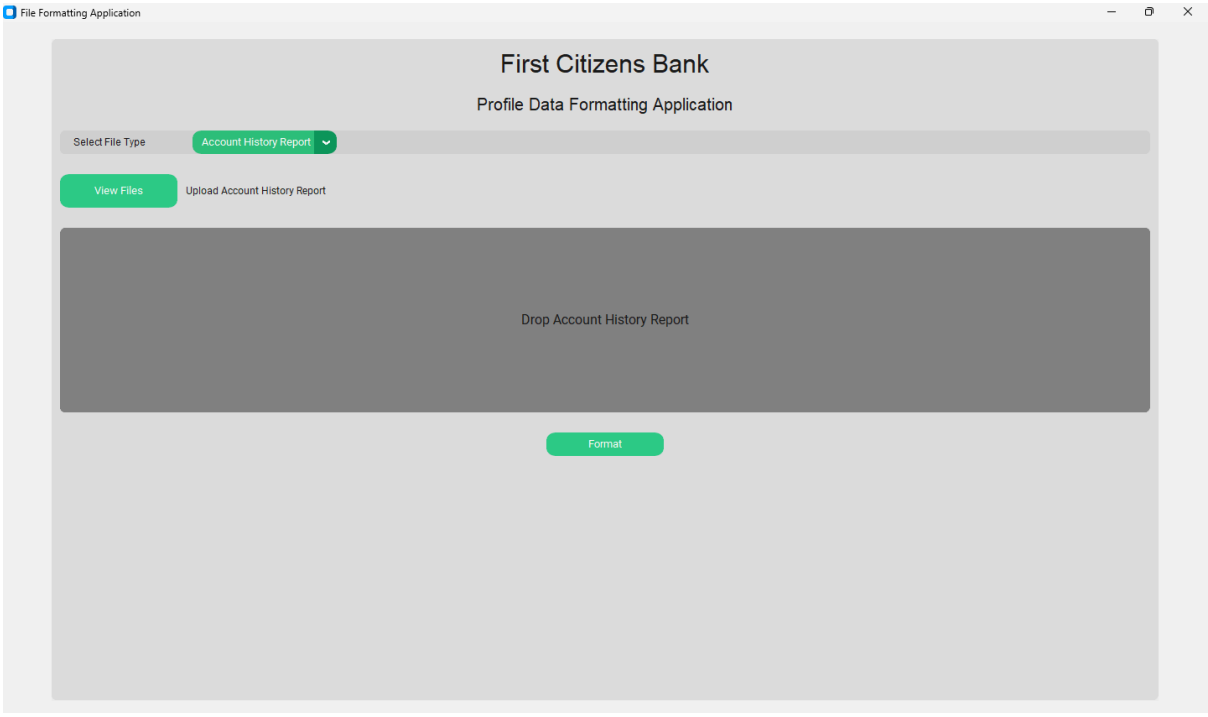


Figure 35: File type selected | User prompted to upload the files

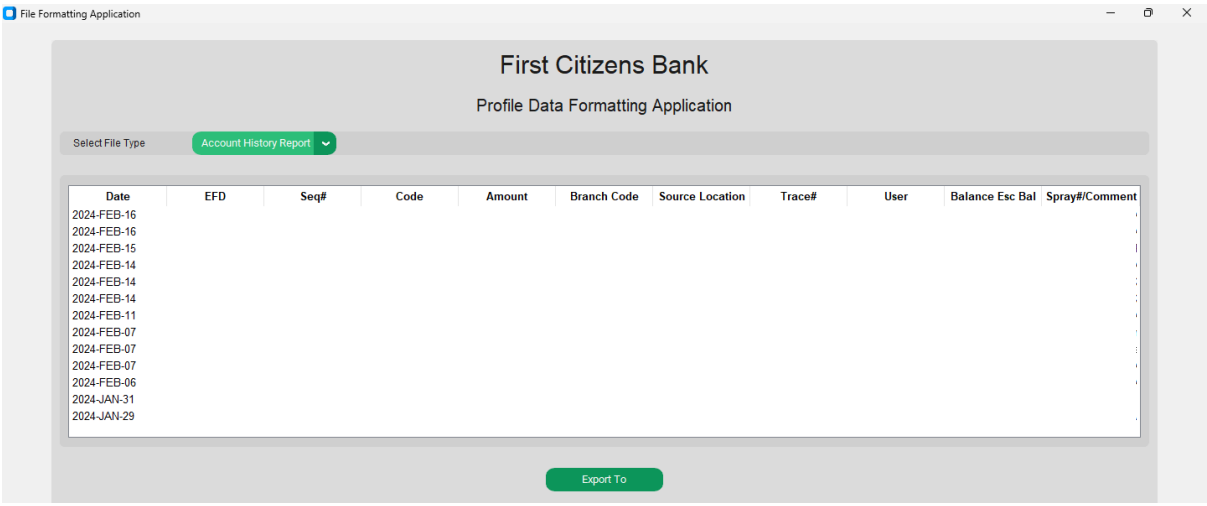


Figure 36: Preview of formatted file

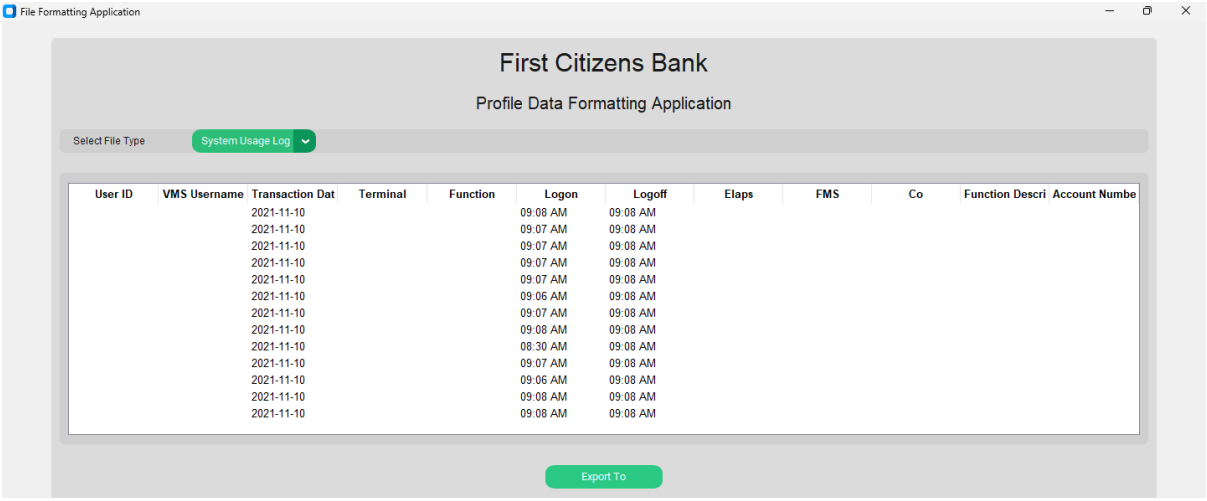


Figure 37: Preview of formatted file

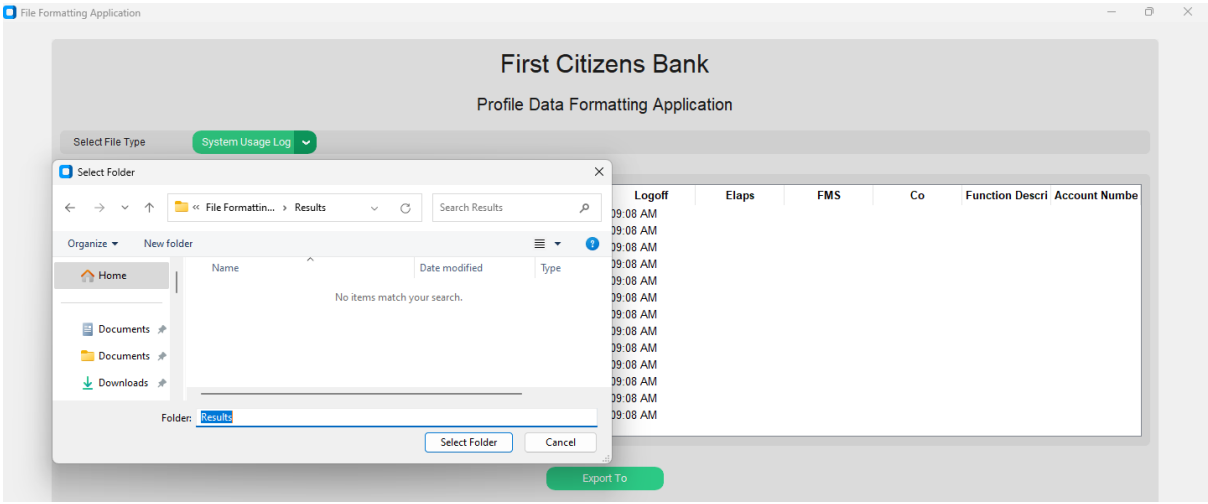


Figure 38: Export To button clicked | File explorer to select destination directory

Figure 39: Export Feedback

THE PROCESS

The user selects from the dropdown menu the type of file they wish to format. The can the either use the View Files button to summon a file explorer or drag and drop from an open file browser the file they wish to format. Once the file path is set the user can click format which reads the file and formats it correctly. The program then displays the formatted file to the user and the user may select the Export To button to select the directory that the formatted file should go to. The formatted file takes the original file name with (formatted) in parentheses for easy identification. The user can then just reselect a file type from the drop down menu to upload and format a new file.

```
@staticmethod
def is_date(string):
    # Regular expression pattern for YYYY-MMM-DD format
    date_pattern = r'^\d{4}-[A-Z]{3}-\d{2}$'
    return bool(re.match(date_pattern, string))
```

Image 4: Regex to determine the start of a new line | new lines start with dates

```
        for line in lines:
            if line.strip() and not re.match(r'^\d{4}-[A-Z]{3}-\d{2}', line.strip()):
                current_record += "  " + line.strip()
            else:
                if current_record:
                    combined_lines.append(current_record.strip())
                current_record = line.strip()

        # Append the last record if there's any remaining
        if current_record:
            combined_lines.append(current_record.strip())

        # Convert the combined lines to a DataFrame
        cleaned_data = "\n".join(combined_lines)
        io_module = io.StringIO(cleaned_data)
```

Image 5: This for loop determines whether the current line starts with a date

If the line does not start with a date that means it is data that has spilled over from the first line above that begins with a date. Spilled over data simply gets concatenated to the end of the line it should be on, separated by two spaces.

```
        df.columns = colnames
        split_columns = df['Spray#/Comment'].str.split(r'\s{2,}', expand=True).fillna('')

        if(split_columns.shape[1] == 2):
            split_columns.columns = ['Spray#', 'Source Location 2']

        elif(split_columns.shape[1] == 3):
            split_columns.columns = ['Spray#', 'Source Location 2', 'Comment']

        elif(split_columns.shape[1] == 4):
            split_columns.columns = ['Spray#', 'Source Location 2', 'Comment', 'Comment2']

        # Concatenate the new columns with the original DataFrame
        df = pd.concat([df.drop(columns=['Spray#/Comment']), split_columns], axis=1)
```

Image 6: Code to separate columns

The two space separation we used to concatenate the data allows us to use the regex pattern r'\s{2,}' to determine the beginning of a different column. Depending on the amount of lines that were spilled over the number of columns we would need to separate would differ as the number of concatenations would have differed, we use a simple conditional branch to determine the number of columns.

```
if(split_columns.shape[1] == 2):
    df['Spray#/Comment'] = df['Spray#'].fillna('') + ' ' + df['Spray#/Comment2'].fillna('')
    df['Source Location'] = df['Source Location'].fillna('') + ' ' + df['Source Location 2'].fillna('')

elif(split_columns.shape[1] == 3):
    df['Source Location'] = df['Source Location'].fillna('') + '  ' + df['Source Location 2'].fillna('')
    df['Spray#/Comment'] = df['Spray#'].fillna('') + ' ' + df['Spray#/Comment2'].fillna('') + ' ' + df['Comment'].fillna('')
    df.drop(columns={'Comment'}, inplace=True)

elif(split_columns.shape[1] == 4):
    df['Source Location'] = df['Source Location'].fillna('') + '  ' + df['Source Location 2'].fillna('')
    df['Spray#/Comment'] = df['Spray#'].fillna('') + ' ' + df['Spray#/Comment2'].fillna('') + ' ' + df['Comment'].fillna('') + ' ' + df['Comment2'].fillna('')
    df.drop(columns={'Comment2', 'Comment'}, inplace=True)
```

Image 7: Joining of split columns

After splitting the columns we can concatenate the correct column to the correct field. Again a conditional branch is used depending on which columns exist in the dataframe.

For the other file type accepted by the program, the solution for formatting spilled over data is comparatively more trivial as the content of the file would only spillover onto the line directly below it.

```
df = pd.read_csv(cleaned_data_io, names=columns, sep=r"\s{2,}", engine='python')

reshaped_data = []
for i in range(0, len(df), 2):
    row1 = df.iloc[i].values
    row2 = df.iloc[i+1].values
    new_row = [row1[0], row2[0]] + list(row1[1:]) + list(row2[1:])
    reshaped_data.append(new_row)

df = pd.DataFrame(reshaped_data)
```

Image 8: For loop to reformat data

The for loop can simply iterate over the dataframe and append the values on the second row to those on the first. The loop would have a step value of two as it would only need to iterate over half of the dataframe as operations on two rows (the row above to append to and the row below which contains the data for appending) are done within each iteration.

DISCUSSION

CONTINUOUS AUDITING PROJECT

The Continuous Auditing Project was the most exhaustive of all the projects undertaken during the 10 week period, beginning in week two (2) and ending in week eight (8). The project encompassed a complete reverse engineering of an existing program written in ACL, and a mimic of the functionality provided by the Excel macros.

The operations performed in ACL were numerous and rather complex, the decision to reverse engineer the script as opposed to tailoring it to fit the new requirement specifications was primarily a decision that can be attributed to the performance enhancement that would be provided by a Python solution. The ACL script performed many operations involving reading and writing to memory, as that is how tables are created and stored in ACL, these read and write operations to disk meant a noticeable performance penalty would be incurred. Simply storing the data in dataframe variables and avoiding those taxing read and write operations reduced the application's runtime significantly.

Upon discussions with the Banking and Support team we outlined additional requirements to perform filtering and aggregation on the data. Filtering based on the types of transactions not subject to the review requirements and aggregating debit transactions and credit transactions separately. Our approach to complete this project was a scrum approach with four (4) separate sprints lasting approximately 29 days when factoring user acceptance testing and implementation.

Sprint cycle 1	10 days
Sprint cycle 2	1 day
Sprint cycle 3	8 days
Sprint cycle 4	10 days

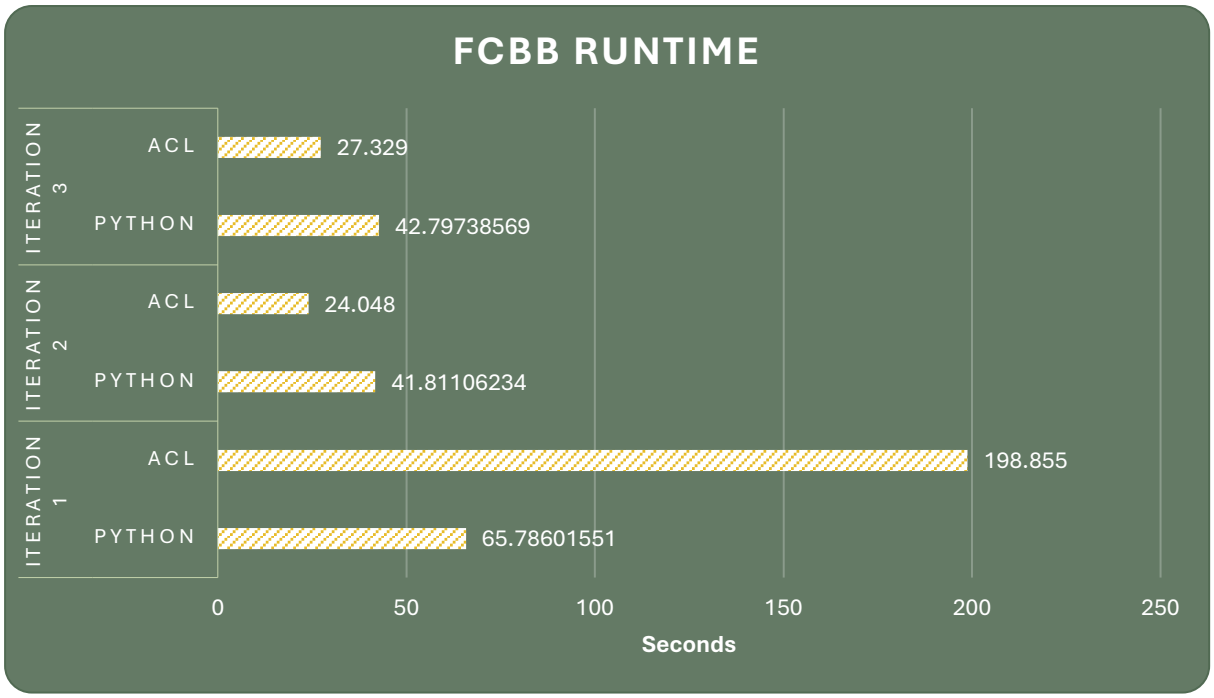


Chart 1: Barbados Application Runtime

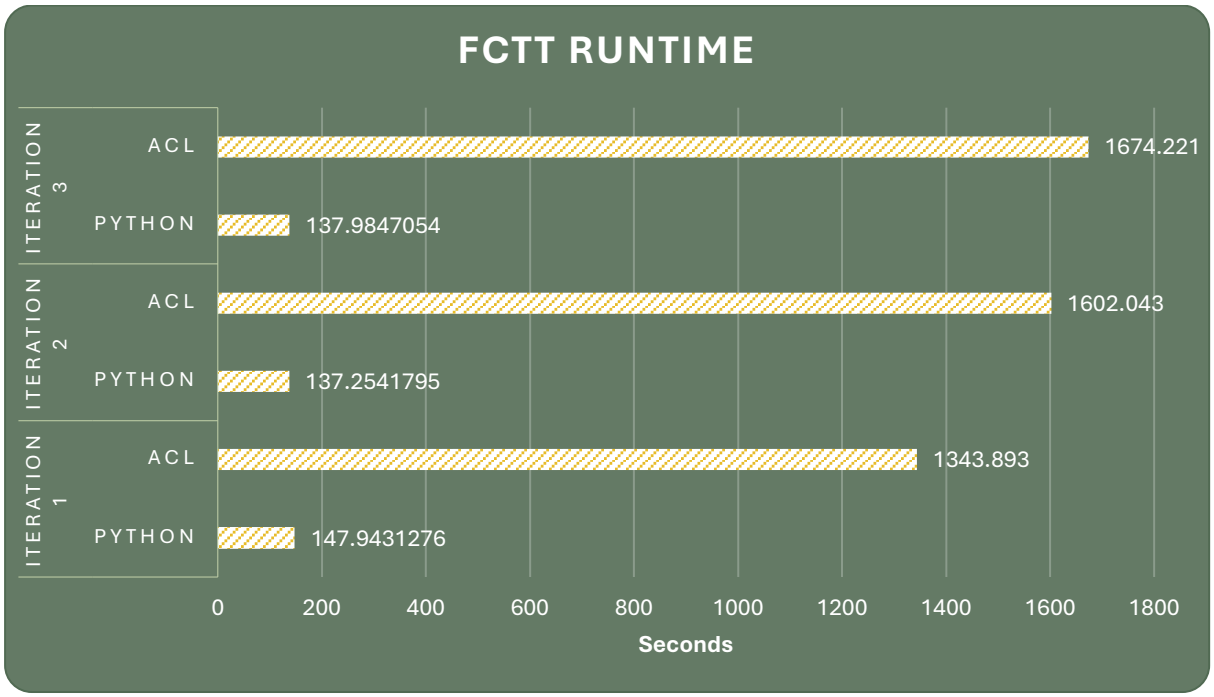


Chart 2: Trinidad and Tobago Application Runtime

This comparison outlines the runtime for the Barbados application and Trinidad and Tobago application compared to the ACL program. Also keep in mind the ACL program is step 1 in a multi-step process, whereas our application performs the entirety of the process.

USER ACCESS VERIFICATION PROJECT

The user access verification project was comparatively simpler to develop than our first project, it began in week eight (8) and ended in week nine (9). The application merely performs a simple mapping of system access to transactions and also obtains the exceptions (accesses without a transaction mapped to it).

Upon discussing the requirements of the application with the investigations team and observing demonstrations of how the manual process is done, it became apparent that for each type of search a very similar series of operations would be done to perform the mapping. Intuiting this, it became obvious that a dropdown menu in the GUI with a class for each search type was the ideal approach to solving the task.

Each menu item summons a different class for that search type. This simple yet elegant solution utilised similar operations as those in our previous project, hence the completion time for this activity was much shorter. We utilised a scrum approach for this application's development process, divided into three (3) sprints with a completion time of approximately 5 days.

Sprint cycle 1	1 day
Sprint cycle 2	1.5 days
Sprint cycle 3	2.5 days

FILE FORMATTING PROJECT

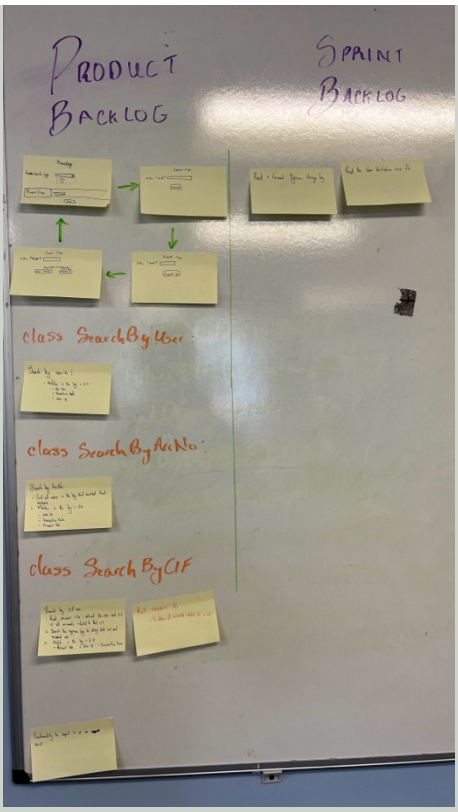
The file formatting project was perhaps the most rudimentary of all projects undertaken during this period, beginning in week nine (9) and ending at the start of week (10). The application quite simply takes text files as input, formats it and exports the formatted data in an Excel format. The cleaning of this data will ideally make the task of making sense of that data easier. The excel format would also allow for the user to utilise Excel's plethora of functions available to view and manipulate the data, a capability not facilitated with data stored in a text file.

The task while seemingly unassuming at first involved some relatively complex logic to actually properly format the data. The data sets contained rows which had values beneath it in subsequent rows and this was the case for multiple columns in each row.

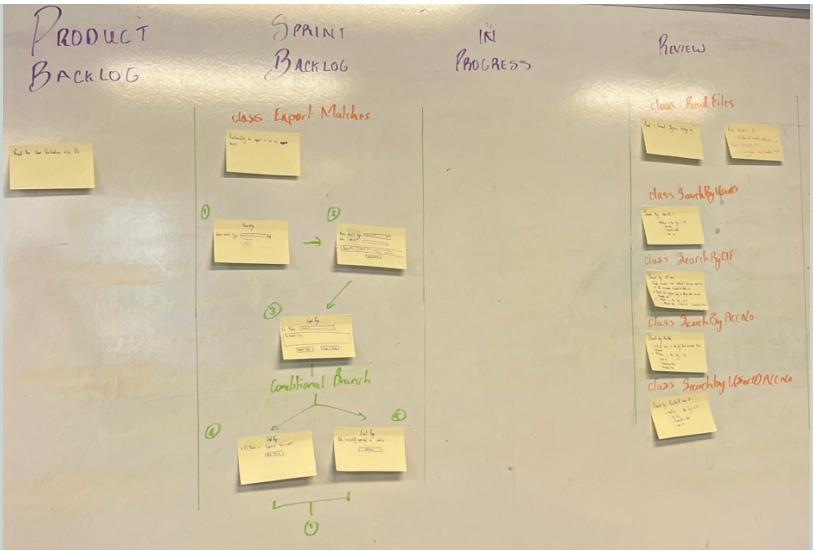
The task was to shift the data that was below it, adjacent to it instead. It involved skipping header information and then reading all rows, and performing string concatenations of values in the rows below to the rows on top, until the value in of a regular expression was matched (signalling the start of a new top row). It was particularly difficult when dealing with data which had an undetermined amount of rows spilling over (sometimes none, sometimes 1 or 2 or 3). Our approach to complete this project also utilised scrum with just two (2) sprints and a completion time of about 3 days.

Sprint cycle 1	1 day
Sprint cycle 2	2 days

SCRUM BOARD



Scrum board for the user access verification project. GUI sketch, description of how data should be mapped, class delineations.



Scrum board of user access verification project in progress with search classes completed and GUI development in the app's development pipeline.

Scrum board showing the file formatting project, a GUI sketch and the file types to format.

External libraries and APIs were needed to perform the complex operations required by the various applications:

- Pandas
- Customtkinter
- Tkinter
- Tkinterdnd2

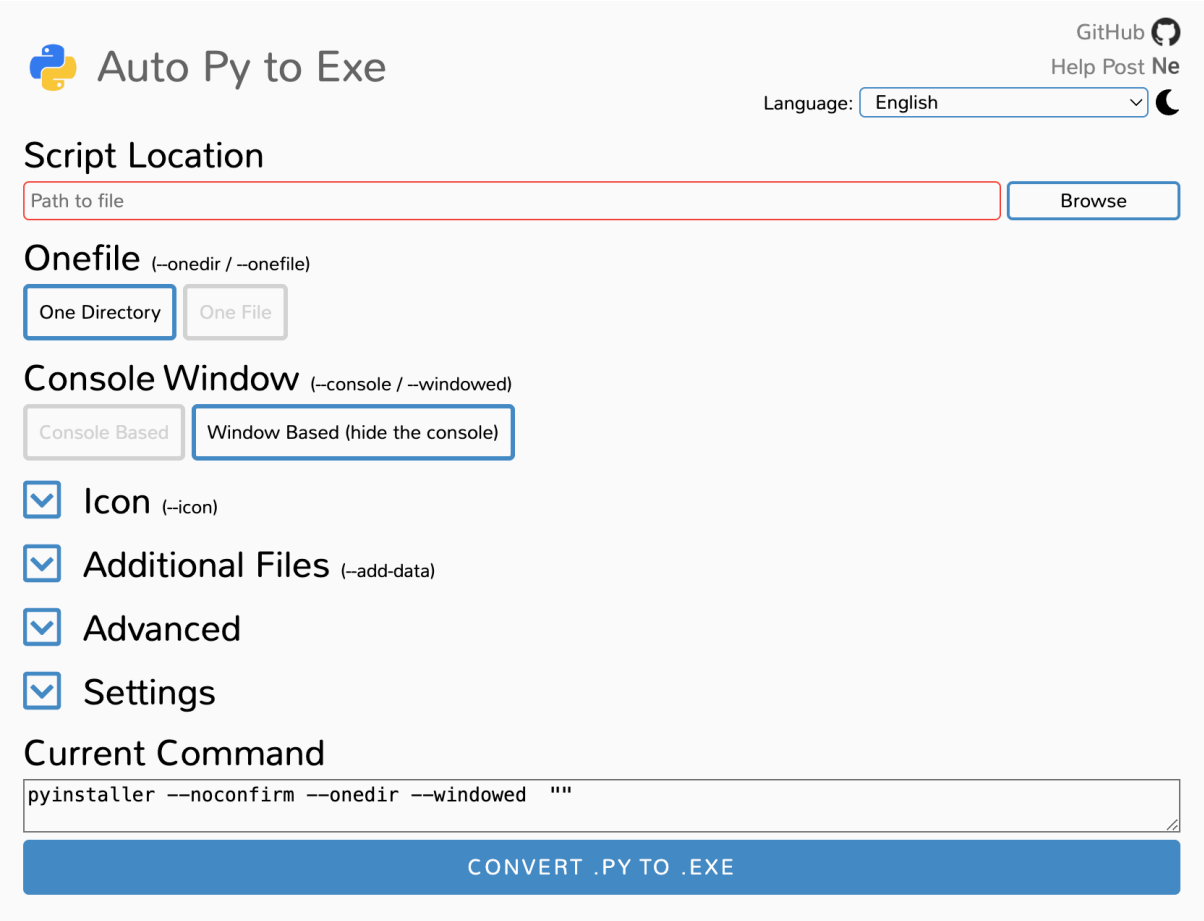
Pandas permitted us to store tabular data in a data structure known as a dataframe, consisting of rows and columns like any other table. Additionally the Pandas library contained methods to perform the merging, subtotalling, sorting and grouping of data. We were also able to perform different types of merge operations:

- Inner merges
- Outer merges
- Left merges
- Right merges

Customtkinter and Tkinter are the backbone of the User Interface, customtkinter provided modern GUI elements whilst Tkinter inter was used to provide GUI elements not supported by customtkinter. Both were crucaial in building the simple yet intuitive user interfaces. Tkinterdnd2 was used in the file formatting application to allow for the drag and drop functionality for file uploading

Full API documentation can be found linked in the **Appendix**.

To package the applications we used auto_py_to_exe a GUI based version of pyinstaller to convert the Python program to an executable file packaged with the applications' dependencies.



This allowed for the applications to run on any machine with a Windows based operating system without needing to interact with an IDE and with the underlying code abstracted away.

CONCLUSION

CONTINUOUS AUDITING PROJECT

The continuous auditing process was fragmented, tedious, slow and contained a substantial amount of data not subject to be reviewed. Our application combined the disjointed processes in one solution, improved the runtime of processing the data, reduced the dataset of the output and made the process more intuitive and user friendly with a User Interface.

USER ACCESS VERIFICATION PROJECT

The user access verification activity entailed a manual and tedious search through text data to determine user accesses and the corresponding reasoning for that access. The data was convoluted and exhaustive to sift through and the process was prone to human error. Our solution automatically mapped accesses to transactions and obtained accesses without transaction mappings automatically as well. It allowed for multiple types of search keys and for the user to enter one or numerous search keys in the search field. The solution also allowed for exporting the data in an easily readable excel format.

FILE FORMATTING PROJECT

Files containing such crucial data in a text file format were difficult to read and poorly structured. Our formatting application restructured the source files and exported the files in a much more useful Excel file format. The User Interface is intuitive and easily navigable and the time savings provided by having properly formatted data is immense.

APPENDIX

Pandas	https://pandas.pydata.org/docs/reference/index.html
Customtkinter	https://customtkinter.tomschimansky.com/documentation/
Tkinter	https://docs.python.org/3/library/tkinter.html
Tkinterdnd2	https://tkinterdnd.sourceforge.net/TkinterDnD.html