UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Faculty of Computing

Software Development I Coursework Report

Module : 4COSC006C.2 Software Development I

Student ID : w2084774/ 20230297

Student First Name : Sivananthan

Student Surname : Anusigan

Date of submission : 29th April 2024

Tutorial group : O

I. Acknowledgment

I hereby confirm my understanding of the concepts of plagiarism, collusion, and contract cheating, having thoroughly acquainted myself with the section on Assessment Offences as detailed in the Essential Information for Students. All materials presented in this submission are original and solely my own.

Additionally, I would like to extend my heartfelt thanks to my 4COSC006C.2 Software Development I lecturer and module leader, Mr. Guhanathan Poravi sir, and to my tutorial lecturers, Mr. Torin Weerasinghe sir and Ms. Kavya Atapattu miss , who gave me guidance and support to complete the coursework with excellence.
Thank you for your support and guidance throughout this process.

II. Table of Contents

III. Table of Figures

1.Design Decisions

For creating the Graphical user interface for the finance tracker Tkinter python library is used. I have made several decisions in arranging the widgets and frames for the GUI. I implemented these in the create_widgets() method in the Finance tracker GUI class. The decisions are;

- Create suitable frames to place the tree view table, Search bar, Scroll bar.
- Position search bar and button at the top of the GUI in the respective frame, where search button is adjacent to the search bar.
- Implement tree view table using “ttk.Treeview” widget and display the transaction details .
- Initialize 4 columns in the tree to display the category, amount, type ,date of transactions.
- Create and position the scrollbar in the right side of the tree in vertical manner.
- Implementation of sort functionality in table columns for the user to sort list transactions.
- Employ a distinct background colour “cadet blue” as the background of the GUI using “ttk.style”.
- Configure font style, font size, other visual properties with the usage of “ttk.style”
- Pack frames in proper manner on which GUI run smoothly.

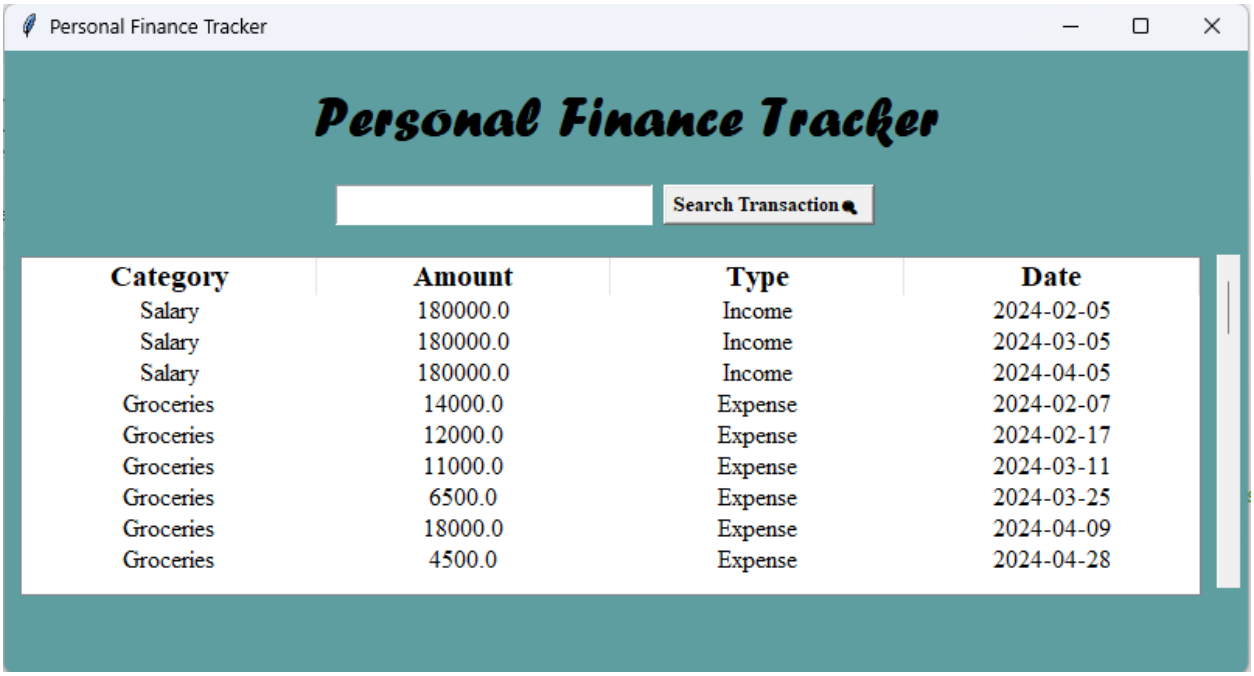


Figure 1-Interface structure of the personal finance tracker

2.Class Structures

"FinanceTrackerGUI" is the main class of the GUI application consisting of methods __init__(self, root), create_widgets(self), load_transactions(self, filename), display_transactions(self, transactions), search_transactions(self), sort_by_column(self, col, reverse) facilitates the development and management of the personal finance tracker of the application.

Methods inside the class and their functions

- __init__(self, root) : Initializing the GUI window and set title for the application, Calls the create_widgets() method and loads transactions from the json file.
- create_widgets(self): Creating and styling all the widgets required for the GUI such as frames, heading, button, tree view table, scrollbar.
- load_transactions(self, filename): Loading transactions from a JSON file and returns them as dictionary.
- display_transactions(self,transactions): Displaying transactions in the treeview table based on the provided transaction data.
- search_transactions(self): Searches the transaction based on user input and displays the search results in the tree view table.
- sort_by_column(self, col. reverse): Sorts the transactions displayed in the tree view

3.GUI Functionality

1. **Search Transactions**
    - Users can input search queries into the provided search bar.
    - The application filters transactions based on the search query, searching across all transaction attributes.
    - Transactions matching the search query are displayed in the treeview, allowing users to quickly locate relevant transactions.

2. **Sorting Transactions**
    - Users can sort transactions displayed in the treeview by clicking on column headers.
    - Clicking on a column header toggles between ascending and descending sorting order.
    - Transactions are sorted based on the selected column, such as category, amount, type, or date, facilitating easy organization and analysis of financial data.

3. **Treeview Display**
    - The treeview widget presents financial transactions in a structured tabular format.

- Each transaction is represented as a row in the treeview, with columns for attributes like category, amount, type, and date.
- Users can scroll through the treeview to view all transactions and interact with individual transaction entries.

4.Source Code

4.1 Source code for GUI

```
#Importing tkinter module to create GUI and ttk for styling and importing json module to load transactions
import tkinter as tk
from tkinter import ttk
import json

#Defining a class for finance tracker graphical user interface
class FinanceTrackerGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Personal Finance Tracker")
        self.create_widgets()
        self.transactions = self.load_transactions("transactions.json")

    def create_widgets(self):
        """Method to create widgets for the GUI"""

        #Frame for table and scrollbar
        #Window Size initialization
        self.root.geometry("800x400")

        #Creating the frame
        self.frame=ttk.Frame(self.root)

        #Creating a style object
        style = ttk.Style()

        #Configuring and applying frame style
        style.configure("Custom.TFrame", background="cadetblue")
        self.frame = ttk.Frame(self.root, style="Custom.TFrame")

        #Packing the frame to the window
        self.frame.pack(fill=tk.BOTH, expand=True)
```

```
        #Adding the heading and packing it to the frame
        heading = tk.Label(self.frame, text="Personal Finance Tracker", font=("Forte", 27,
"bold"),background="cadetblue")
        heading.pack(pady=(20,0))

        #Adding and packing a vertical scroll bar at the right of the frame
        scrollbar = tk.Scrollbar(self.frame, orient="vertical",width=15)
        scrollbar.pack(side=tk.RIGHT,fill=tk.Y, padx=(0, 5), pady=(65,55))



        #Search bar and button
        #Creating a frame to place search entry bar and search button and packing it in the top
center of the window
        search_container = tk.Frame(self.frame, background="cadetblue")
        search_container.pack(side=tk.TOP, anchor=tk.CENTER, pady=20)

        #Creating StringVar to store search query
        self.search = tk.StringVar()

        #Creating an entry widget for the search query and packing it
        search_entry = tk.Entry(search_container,text=self.search, width=20, font=("Times New
Roman", 15))
        search_entry.pack(side=tk.LEFT)

        #Creating the search button and enabling search function to it
        s_button = tk.Button(search_container, text="Search Transaction 🔍 ", font=("Times New
Roman", 10, "bold"), command=self.search_transactions, width=18)
        s_button.pack(side=tk.LEFT, padx=7)



        # Treeview for displaying transactions
        #Creating treeview widget with necessary columns
        self.tree = ttk.Treeview(self.frame, columns=("Category", "Amount", "Type", "Date"),
show="headings",yscrollcommand=scrollbar.set)

        #Configuring the style of the treeview and the style of the heading
        style.configure("Treeview", font=("Times New Roman", 12),background="White")
        style.configure("Treeview.Heading", font=("Times New Roman", 14, "bold"))

        #Defining the column names of the treeview
        self.tree.heading("Category", text="Category", command=lambda:
self.sort_by_column("Category", False), anchor="center")
        self.tree.heading("Amount", text="Amount", command=lambda:
self.sort_by_column("Amount", False), anchor="center")
```

```
        self.tree.heading("Type", text="Type", command=lambda: self.sort_by_column("Type",
False), anchor="center")
        self.tree.heading("Date", text="Date", command=lambda: self.sort_by_column("Date",
False), anchor="center")

        #Setting column widths and alignment
        self.tree.column("Category", width=100, anchor="center")
        self.tree.column("Amount", width=100, anchor="center")
        self.tree.column("Type", width=100, anchor="center")
        self.tree.column("Date", width=100, anchor="center")

        self.tree.pack(fill=tk.BOTH,expand=True, padx=10, pady=(0,50))

        # Configuring the scrollbar to scroll the treeview when scrolling
        scrollbar.config(command=self.tree.yview)



    def load_transactions(self, filename):
        """Method to Load transactions from the JSON file"""
        try:
            #Loading transactions form the JSON file "transactions.json"
            with open(filename, "r") as file:
                transactions = json.load(file)
            #Returning the loaded transactions
            return transactions
        except FileNotFoundError:
            #Returning and empty dictionary if file is missing
            return {}

    def display_transactions(self, transactions):
        """Method to display transactions in the tree table"""

        #Remove existing entries
        for item in self.tree.get_children():
            self.tree.delete(item)

        # Add transactions to the treeview

        #Getting the data such as category,date,amount,transaction type from the dictionary
        for category,transaction_data in transactions.items():
            for transaction in transaction_data:
                date = transaction.get("date")
                amount = transaction.get("amount")
                transaction_type = transaction.get("type")
```

```
            # Inserting the transaction data into the tree view
            self.tree.insert("", "end", values=(category, amount, transaction_type, date))

    def search_transactions(self):
        """Method to search transactions by any attribute of the transaction"""
        #Getting search query from user and making it to simple letters
        search = self.search.get().lower()
        if search:
            output = {}
            #Iterating over all the category and transactions belongs to it
            for category, transaction_data in self.transactions.items():
                for transaction in transaction_data:
                    #Checking whether the search query matches with the category
                    if search in category.lower():
                        #If not matching with category initializing a list for category
                        if category not in output:
                            output[category] = []
                        #Add transaction to the category's list if it matches
                        output[category].append(transaction)
                    else:
                        # Checking whether the search query matches any value in the transaction
                        for value in transaction.values():
                            if search in str(value).lower():
                                if category not in output:
                                    output[category] = []
                                output[category].append(transaction)

                                #Breaking the loop after finding a match to avoid duplicates
                                break
            #Displaying the search results in the tree view
            self.display_transactions(output)
        else:
            #Displaying all the transactions if search query is empty
            self.display_transactions(self.transactions)



    def sort_by_column(self, col, reverse):
        """Method to sort columns in ascending or descending order"""

        if col == "Category":
            #Sorting the categories in alphabetical order
            categories = sorted(self.transactions.keys(), reverse=reverse)

            #Getting the current position of each category in the tree
```

```
            children = {child: self.tree.set(child, "Category") for child in self.tree.get_children()}
            index = 0

            #Moving each category to their sorted position through iteration
            for category in categories:
                #Iterating across each child item and its respective category values in the treeview
                for child, cat_value in children.items():
                    #Checking whether the category value of the child matches the current category in
process
                    if cat_value == category:
                        #If yes, moving category to the new position
                        self.tree.move(child, "", index)
                        #Increasing the index to reflect the new position for next upcoming category
                        index += 1

        elif col == "Amount":
            #Creating a list of tuples where each tuple contains the amount value of a transaction and
its corresponding child item in the treeview  to sort amount
            value= [(float(self.tree.set(child, col)), child) for child in self.tree.get_children("")]
            value.sort(reverse=reverse)
            index = 0

            #Moving each transaction to its sorted position
            for data, child in value:
                self.tree.move(child, "", index)
                index += 1
        else:
            #Sorting for transaction type and transaction date
            value=[(self.tree.set(child, col), child) for child in self.tree.get_children("")]
            value.sort(reverse=reverse)
            index = 0
            for data, child in value:
                self.tree.move(child, "", index)
                index += 1
        #Set sorting indicator for the sorted column
        if reverse:
            sorting_indicator = "↓"
        else:
            sorting_indicator = "↑"

        column_heading = f"{col} {sorting_indicator}"

        #Updating the column heading in the treeview to indicate the sorting direction
        self.tree.heading(col,text=column_heading)

        # Toggling the sorting direction for next click on column heading
```

```
        self.tree.heading(col, command=lambda: self.sort_by_column(col, not reverse))


def main():
    root = tk.Tk()
    app = FinanceTrackerGUI(root)
    app.display_transactions(app.transactions)
    root.mainloop()

if __name__ == "__main__":
    main()
```

4.2 source code to intergrate coursework part C on coursework part B

```
#Importing json module to deal with data
import json

#Importing the 'main' function from the 'w2084774_part_C' module
from w2084774_part_C import main

#Global dictionary to store transactions
transactions = {}

#File handling functions
def load_transactions():
    """Function to load data from the JSON file transactions.json"""
    global transactions
    try:
        with open("transactions.json", "r") as file:
            transactions = json.load(file)
    except FileNotFoundError:
        """if such file is not found create an empty dictionery transactions"""
        transactions = {}

def save_transactions():
    """Function to create a JSON file transactions.json and save the transactions in dictionery to
the JSON file"""
    global transactions
    with open("transactions.json", "w") as file:
        # Write the transactions dictionary to the file using json.dump()
        json.dump(transactions, file, indent=2)

def read_bulk_transactions_from_file(filename):
    """Function to read bulk transaction data from a text file and add them to the dictionery"""
    global transactions
```

```
    try:
        with open(filename, "r") as file:
            # Reading all the lines in the file
            lines = file.readlines()
            for line in lines:
                # Splitting each line
                data = line.strip().split(",")
                # Checking the length of data to ensure all necessary data are present
                if len(data) == 4:
                    # Extracting transaction data from line
                    t_amount, t_category, t_type, t_date = map(str.strip, data)

                    # Validate transaction amount in the text file
                    try:
                        t_amount = float(t_amount)
                    except ValueError:
                        # Printing an error message if the transaction amount is invalid
                        print(f"In {line.strip()}: {t_amount} is an invalid transaction amount. Transaction
omitted.")
                        continue  # To skipping to the next iteration

                    # Validate transaction type in the text file
                    if t_type.capitalize() != "Income" and t_type.capitalize()!= "Expense":
                        # Printing an error message if the transaction type is not (Income/Expense)
                        print(f"In {line.strip()}: {t_type} is an invalid transaction type. Transaction
omitted.")
                        continue  # To skipping to the next iteration

                    # Validating the transaction date format in the text file
                    if t_date[4]!="-" or t_date[7]!="-":
                            #printing an error message if the transaction date is invalid
                            print(f"In {line.strip()}: {t_date} is not in YYYY-MM-DD format. Transaction
omitted." )
                            continue  #To skipping to the next itertion

                    # Capitalizing the transaction categories in text file
                    t_category=t_category.capitalize()

                    # Adding the transactions to the global transactions dictionery having transaction
category as key
                    if t_category in transactions:
                        transactions[t_category].append({"amount": t_amount, "type": t_type.capitalize(),
"date": t_date})
                    else:
                        transactions[t_category] = [{"amount": t_amount, "type": t_type.capitalize(),
"date": t_date}]
```

```
            else:
                # Printing an error message if the line does not have the necessary sufficient data
                print(f"In {line.strip()}: Transaction omitted due to insufficient data.")
        # Printing a success message after reading all transactions from the text file
        print("\nBulk transactions read from file successfully")
    except FileNotFoundError:
        # Printing an error message if the file is not found
        print("File is not found")
    # Saving the read bulk transactions to the JSON file
    save_transactions()


# Feature implementations
def add_transaction():
    """Function to add a transaction to the  dictionery"""
    global transactions
    t_amount = 0
    t_category = ""
    t_date = ""

    # Asking the transaction related details from the user as input
    while True:
        # Validating the transaction amount input
        try:
            t_amount = float(input("Enter the transaction amount: "))
            if t_amount <= 0:
                print("Amount must be a positive number. Please try again!")
            else:
                break
        except ValueError:
            # Printing an error message if the user does not give float value
            print("Invalid amount. Please try again!")

    while True:
        t_category = input("Enter the transaction category: ")
        t_category=t_category.capitalize()
        #Printing an error message if the user skip to give transaction category input
        if t_category == "":
            print("Transaction category can't be null. Please try again!")
        else:
            break
    # Validating the transaction type input
    while True:
        t_type=str(input("Enter the transaction type (Income/Expense): "))
        t_type=t_type.capitalize()
```

```
        # Checking whether user enters Income or Expense .If he enters something otherthan this
notifying an error
        if t_type!="Income" and t_type!="Expense":
            print("Invalid input. Transaction type should be Income or Expense")
        else:
            break

    # Validating the transaction date input by checking whether it is in (YYYY-MM-DD) format
    while True:
        t_date = input("Enter the transaction date in YYYY-MM-DD format: ")
        if len(t_date) == 10:
            if t_date[4] == "-" and t_date[7] == "-":
                break
            else:
                print("Invalid input. Enter the date in YYYY-MM-DD format")
        else:
            print("Invalid input. Enter the date in YYYY-MM-DD format: ")

    # Creating a dictionary to store transaction data
    transaction = {"amount": t_amount, "type": t_type, "date": t_date}

    # Appending the transaction to the appropriate category in the transactions dictionery
    if t_category in transactions:
        transactions[t_category].append(transaction)
    else:
        transactions[t_category] = [transaction]

    # Saving the transactions to the JSON file
    save_transactions()

    # Printing a success message after transaction successfully added to the tracker
    print("Transaction added successfully to the tracker")

def view_transactions():
    """Function to display all the transactions categorized by category stored in dictionery"""
    global transactions
    if transactions:
        # Iterates all over each category and its transactions
        for category, category_data in transactions.items():
            # Printing the category name
            print(f"\nCategory: {category}")
            number=1
            # Iterating through each and every transaction in the specific category
            for transaction in category_data:
                # Printing the transaction details
```

```
            print(f"{number}. Transaction amount: {(transaction['amount'])}, Transaction type:
{(transaction['type'])}, Transaction date: {transaction['date']}")
            number += 1

    else:
        # Printing a message if there is no transactions exists
        print("No transactions available")

def update_transaction():
    """Function to allow the user to update the existing added transactions"""
    global transactions

    """Calling the view transactions function to display the existing transactions to assist"""
    view_transactions()
    print("\n")

    # Checking whether transactions exists
    if transactions:
        # Asking for the category to be updated from the user
        category = input("Enter the transaction category to be updated: ")
        category=category.capitalize()

        # Checking whether that category exists
        if category in transactions:
            category_data = transactions[category]

            # Asking the user for transaction index to be updated
            while True:
                try:
                    number = int(input("Enter the transaction number to be updated: ")) - 1
                    if 0 <= number < len(category_data):
                        break
                    else:
                        # Printing an error message if the user enters a non existing index number
                        print("Invalid transaction number. Please try again!")
                except:
                    # Printing an error message if the user enters invalid data type for transaction index
number
                    print("Invalid input. Please try again!")

            # Asking updating transaction details form the user as input
            while True:
                # Asking and validating the transaction amount user enters
                try:
                    t_amount = float(input("Enter the new transaction amount: "))
                    if t_amount <= 0:
```

```
                print("Amount must be a positive number. Please try again!")
            else:
                break
        except ValueError:
            print("Invalid amount. Please try again!")


    while True:
        # Asking and validating the transaction type user enters
        t_type=str(input("Enter the transaction type (Income/Expense): "))
        t_type=t_type.capitalize()
        #Checking whether user enters Income or Expense .If he enters something otherthan
this notifying an error
        if t_type!="Income" and t_type!="Expense":
            print("Invalid input. Transaction type should be Income or Expense")
        else:
            break

    while True:
        # Asking and validating the transaction date user enters
        t_date = input("Enter the new transaction date in YYYY-MM-DD format: ")
        if len(t_date) == 10:
            if t_date[4] == "-" and t_date[7] == "-":
                break
            else:
                print("Invalid input. Enter the date in YYYY-MM-DD format")
        else:
            print("Invalid input. Enter the date in YYYY-MM-DD format: ")

    # Updating the new transaction details in the dictionery
    category_data[number]={"amount":t_amount, "type":t_type, "date":t_date}

    # Saving the updated transaction to the file
    save_transactions()

    # Printing a success message if transaction is successfully updated
    print("Transaction successfully updated")
  else:
    # Printing a message if the category user wants to update is not exists
    print("Category not found.")
else:
  # Printing a message if there is no transactions present in that category for the user to update
  print("No transactions available in that category")

def delete_transaction():
  """Function to allow users to delete transactions present in the financial tracker"""
```

```
    global transactions

    # Calling the view transactions functions to display the transactions present
    view_transactions()
    print("\n")

    # Checking whether there are existing transactions
    if transactions:
        # Asking the user for the category where the transaction to be deleted is present
        category = input("Enter the category of the transaction to be deleted: ")
        category=category.capitalize()

        # Checking if that category exists
        if category in transactions:
            category_data= transactions[category]
            while True:
                # Asking and validating the  index number of the transaction to be deleted
                try:
                    number = int(input("Enter the transaction number to be deleted: ")) - 1
                    if 0 <= number< len(category_data):
                        break
                    else:
                        # Printing an error message if the input index number does not exist
                        print("Invalid Transaction number. Please try again!")
                except:
                    # Printing an error message if user input any other data typen instead of integer for
transaction index number
                    print("Invalid input. Please try again!")

            # Delete the transaction
            del category_data[number]

            # Removing the category if the category becomes empty after the transaction is deleted
            if len(category_data) == 0:
                del transactions[category]

            # Saving the transaction to the JSON file
            save_transactions()

            # Printing a success message if the transaction deleted successfully
            print("Transaction successfully deleted")
        else:
            # Printing a message if there are no transaction exist in the category user entered
            print("Category not found.")
```

```
def display_summary():
    """Function to calculate and display the summary of all transactions in the finance tracker"""
    global transactions
    total_income = 0
    total_expense = 0
    income_categories={}
    expense_categories={}

    # Iterating through the transactions and categorizing them to calculate summary
    for category, category_data in transactions.items():
        for transaction in category_data:

            if transaction['type'] == 'Income':
                total_income += transaction['amount']

income_categories[category]=income_categories.get(category,0)+transaction["amount"]
            elif transaction['type'] == 'Expense':
                total_expense += transaction['amount']

expense_categories[category]=expense_categories.get(category,0)+transaction["amount"]
    # Calculating the net value by subracting total expense from total income
    net_value = total_income - total_expense

    # Foramatting summary values
    total_income=(f"{total_income:.2f}")
    total_expense=(f"{total_expense:.2f}")
    net_value=(f"{net_value:.2f}")



    # Printing the summary of transactions
    head="TRANSACTIONS SUMMARY".center(50,".")
    print("\n",head)

    print("\nIncome Categories")
    print("--------------------")
    if income_categories:
        for category,amount in income_categories.items():
            print(f"{category}:{amount}")
    else:
        print("No income categories")


    print("\nExpense Categories")
    print("--------------------")
    if expense_categories:
```

```
        for category, amount in expense_categories.items():
            print(f"{category}:{amount}")
    else:
        print("No expense categories")

    print("\n*******************************")
    print("Total Income:  ",total_income)
    print("Total Expense: ",total_expense)
    print("Net value:     ",net_value)
    print("*******************************")



def main_menu():
    load_transactions()
    while True:
        print("\nPersonal Finance Tracker")
        print("1. Add Transaction")
        print("2. Read bulk transactions from a file")
        print("3. View Transactions")
        print("4. Update Transaction")
        print("5. Delete Transaction")
        print("6. Display Summary")
        print("7. Personal finance tracker GUI")
        print("8. Exit program")
        choice = input("\nEnter your choice: ")

        if choice == '1':
            add_transaction()
        elif choice == '2':
            filename = input("Enter the file name to read bulk transaction: ")
            read_bulk_transactions_from_file(filename)
        elif choice == '3':
            view_transactions()
        elif choice == '4':
            update_transaction()
        elif choice == '5':
            delete_transaction()
        elif choice == '6':
            display_summary()
        elif choice == '7':
            main()
        elif choice == '8':
            print("Exiting program.")
            break
        else:
```

```
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main_menu()

# if you are paid to do this assignment please delete this line of comment
```

5.Catalog to run the GUI application

1. Download and Install Python 3 version from the official python website.
2. Ensure that tkinter, json packages are included in your python installations.
3. Locate the finance tracker GUI file (w2084774.py) and “transactions.json” file in same folder.
4. Double click the GUI python file to launch the file.
5. Once the application is launched you can view the transaction datas in columns of the treeview.
6. Click any column heading in the treeview to perform sort specific to that column in ascending order. Click again to sort in descending order. Can click repeatedly and toggle the directions and organize transaction details according to your preference.
7. To search for specific transactions, type the keyword or attribute into the search bar provided. And click the **"Search Transaction** 🔍 " to initiate the search. The application will filter transactions based on your search query and display only the matching results in the table.
8. Use the scroll bar in the right side of the interface to scroll through the table to view all transactions.
9. Click on individual rows to select or highlight specific transactions.
10. To exit the application, close the GUI window by clicking the (X) button on the top right corner of the window.

6.Test

6.1 Test plan

- Verify the functionality of search transactions.
- Verify functionality of sort transactions.
- Verifying the functionality of display transactions in treeview.
- Validate the functionality of buttons and scrollbar.
- Ensure smooth program flow and overall functionality.

6.2 Test Cases

Test Case Number	Description	Test Input	Expected Output	Actual Output	PASS/FAIL
1	Loading transactions from json file	“transactions.json”	Should load transactions successfully from the file	Transactions loaded successfully from file	PASS
2	Testing the placement of widgets in GUI	-	All widgets should be successfully created and placed within frames properly.	Widgets created and placed inside the allocated frames successfully	PASS
3	Testing the functionality of buttons	-	Should work properly	Worked Successfully	PASS
4	Testing whether transactions are displayed correctly in the treeview table	transaction data	Should display transaction data properly on respective columns	Transaction data properly displayed on respective columns successfully	PASS
5	Searching for a transaction category with an exact match	Search query=“Groceries”	Should display transactions under category “Groceries”	Transactions under category “Groceries” displayed successfully	PASS
6	Searching for a transaction category with a partial match	Search query= “Sal”	Should display transactions under category name starting with “Sal”	Transactions under category name starting with “Sal” displayed succesfully	PASS
7	Searching for a transaction amount	Search query= “6500”	Should display transactions associated with amount 6500	Displayed transaction associated with a transaction amount 6500 successfully	PASS
8	Searching for a transaction type	Search query=”Income”	Should display transactions associated with transaction type Income	Displayed transactions associated with transaction type Income successfully	PASS
9	Searching for a transaction date	Search query= “2024-04-28”	Should display transactions associated with	Displayed transactions associated	PASS

			transaction date “2024-04-28”	with transaction date “2024-04-28” successfully	
10	Searching for a transaction that does not exist	Search query= “abcd”	Should display a empty tree view table	display a empty tree view table successfully.	PASS
11	Sorting Category column in Ascending order (A-Z)	Click the category column heading once	Transactions should be sorted alphabetically by category from A-Z	Transactions sorted alphabetically by category from A-Z successfully	PASS
12	Sorting Category column in descending order (Z-A)	Click the category column heading twice	Transactions should be sorted alphabetically by category from Z-A	Transactions sorted alphabetically by category from Z-A successfully	PASS
13	Sorting Amount column in Ascending order	Click the Amount column heading once	Transactions should be sorted in ascending order by amount value	Transactions sorted in ascending order by amount value successfully.	PASS
14	Sorting Amount column in descending order	Click the Amount column heading twice	Transactions should be sorted in descending order by amount value	Transactions sorted in descending order by amount value successfully	PASS
15	Sorting Type column in ascending order	Click the Type column heading once	Transactions should be sorted in order by Type Expense followed by Income	Transactions sorted in order by Type Expense followed by Income successfully	PASS
16	Sorting Type column in descending order	Click the Type column heading twice	Transactions should be sorted in order by Type Income followed by Expense	Transactions sorted in order by Type Income followed by Expense successfully	PASS
17	Sorting Date column in ascending order	Click the Date column heading once	Transactions should be sorted in ascending order by Date	Transactions sorted in ascending order by Date successfully	PASS

18	Sorting Date column in descending order	Click the Date column heading twice	Transactions should be sorted in descending order by Date	Transactions sorted in descending order by Date successfully	PASS
19	Testing the functionality of the scrollbar	-	Should scroll across the tree view and work as expected	Scroll worked across the tree view and worked as expected successfully	PASS
20	Integrating with part B coursework and running	Importing main function from the part C module	Should run successfully when it called in part B	ran successfully when it's called in part B	PASS

5.3 Test Summary

- **Number of Tests carried out:** 20
- **Number of Tests Passed:** 20
- **Number of Tests Failed:** 0

Reflection

The testing process has been successful, with all test cases passing without encountering any failures. This indicates that the program performs as expected and meets the specified requirements.

The test cases covered various aspects of the program, including loading transactions, displaying them in the GUI, searching, sorting, and user interaction. No bugs or issues were identified during testing, indicating that the program is robust and stable.

Adjustments Made

Based on the testing results, no adjustments were carried as the program passed all test cases without encountering any failures.

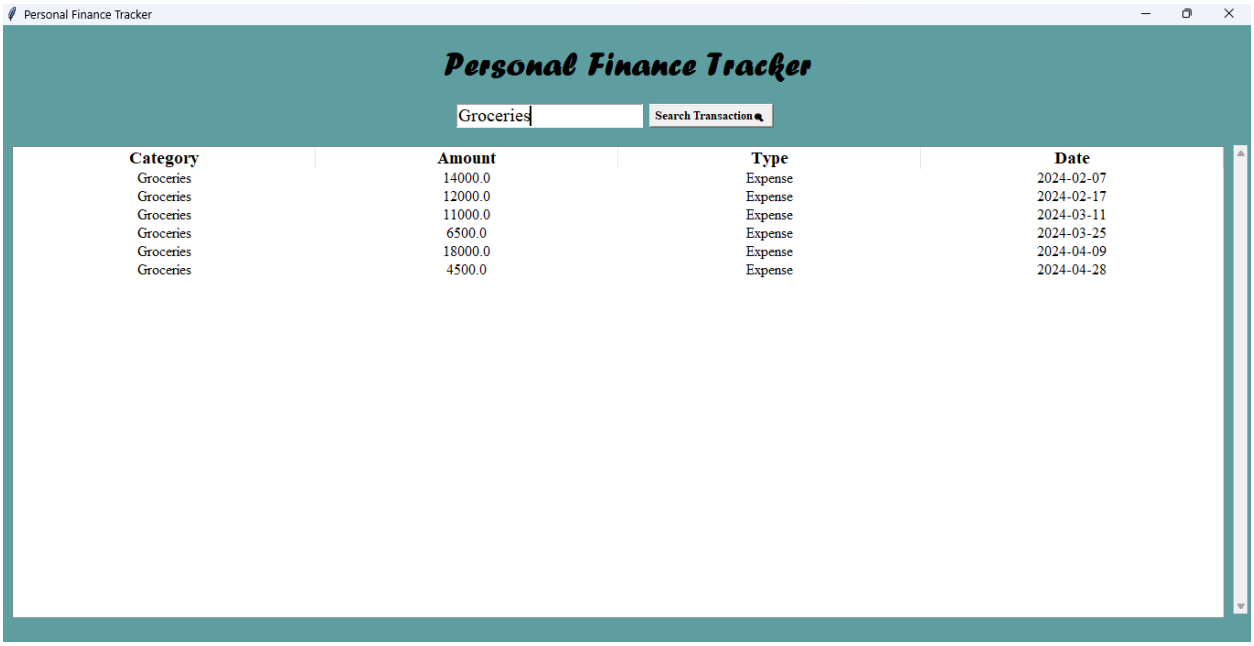


Figure 2-Searching a transaction by category

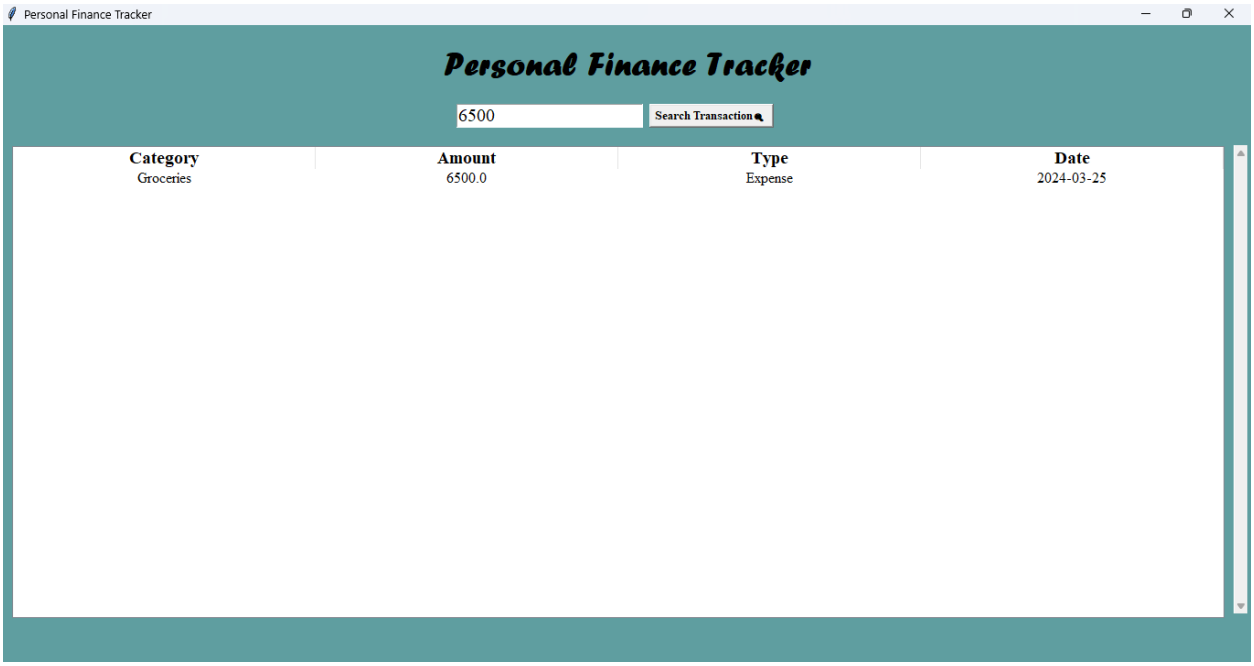


Figure 3-Searching a transaction by amount

Figure 4-Searching a transaction by type

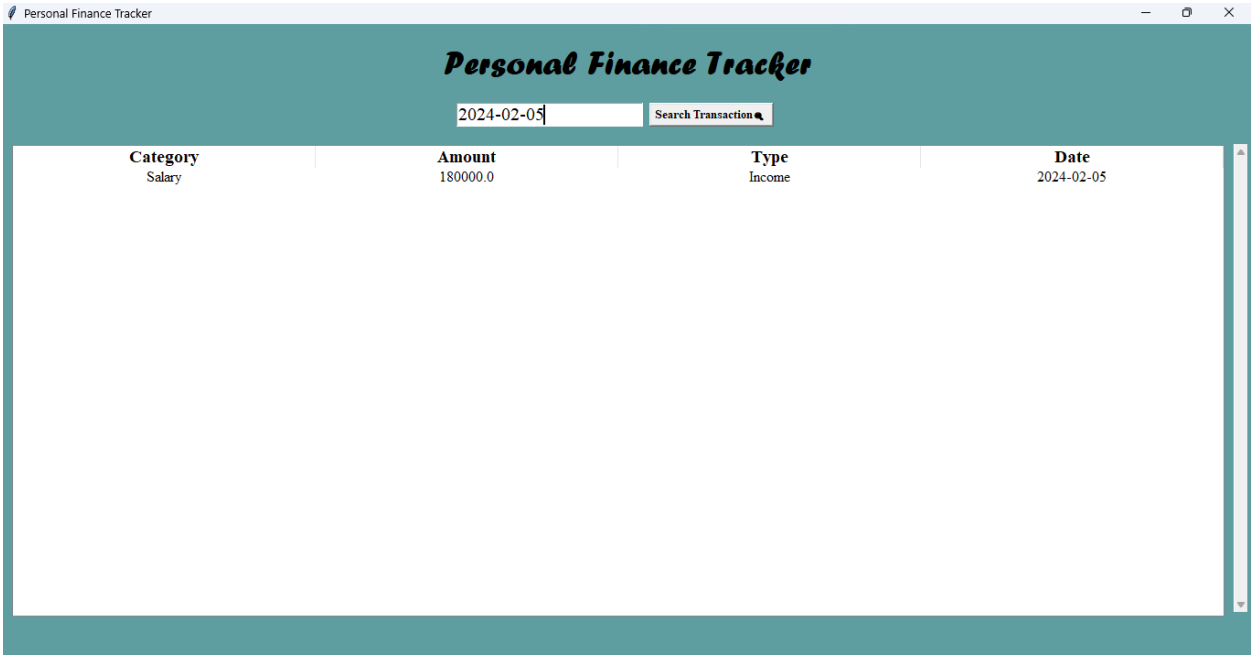


Figure 5- Searching a transaction by date

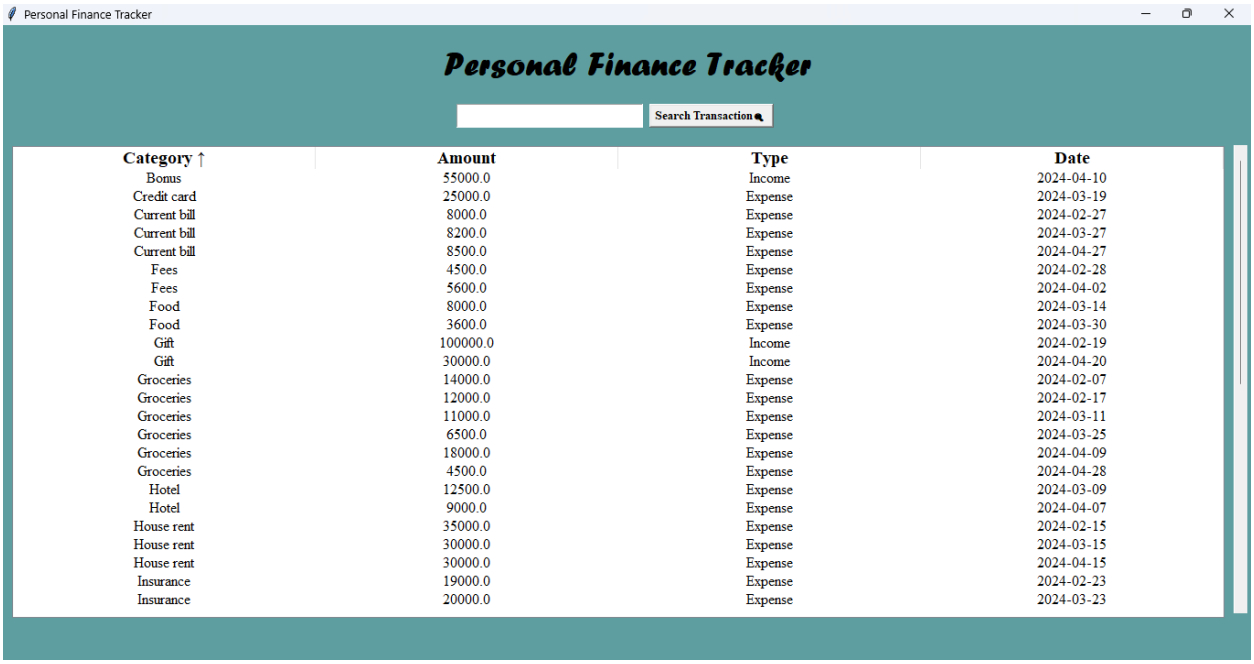
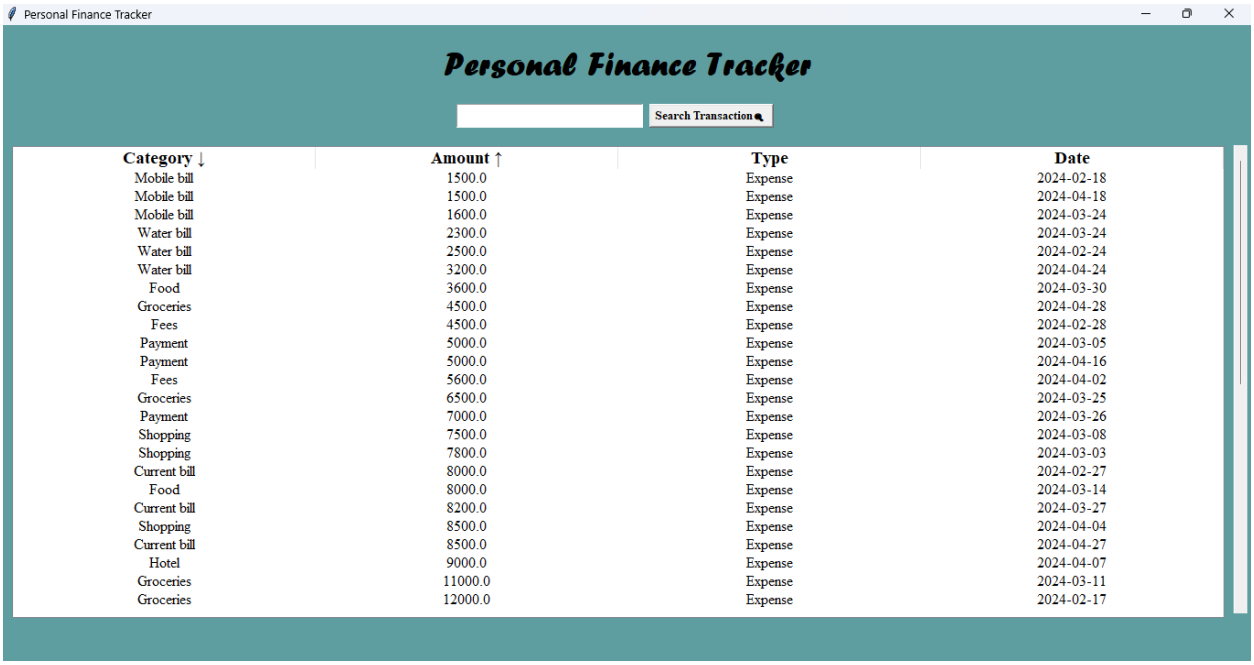


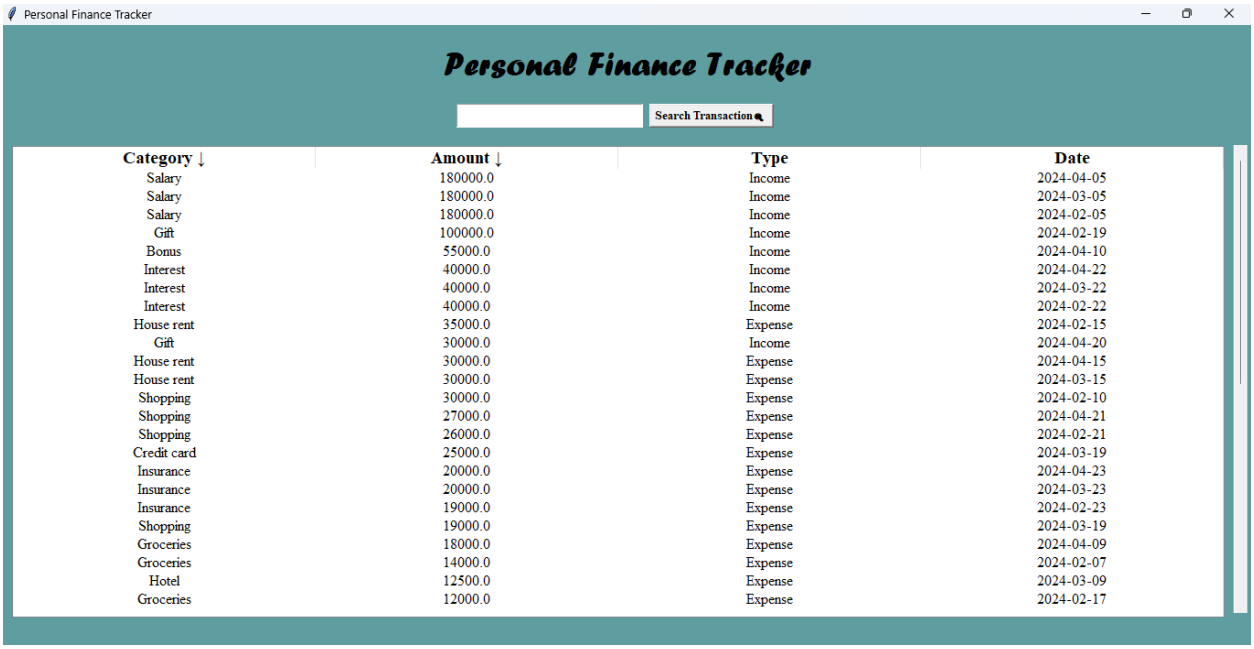
Figure 6-Sorting transaction category in ascending order

Figure 7-Sorting transaction category in descending order



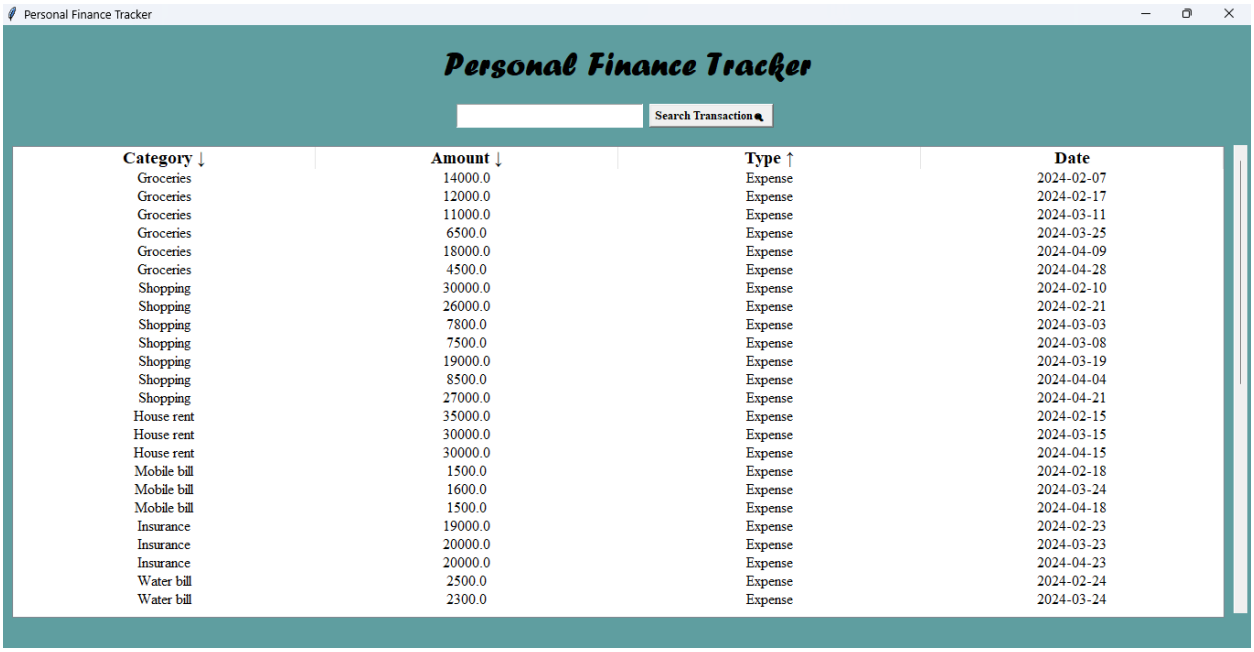
Category ↓	Amount ↑	Type	Date
Mobile bill	1500.0	Expense	2024-02-18
Mobile bill	1500.0	Expense	2024-04-18
Mobile bill	1600.0	Expense	2024-03-24
Water bill	2300.0	Expense	2024-03-24
Water bill	2500.0	Expense	2024-02-24
Water bill	3200.0	Expense	2024-04-24
Food	3600.0	Expense	2024-03-30
Groceries	4500.0	Expense	2024-04-28
Fees	4500.0	Expense	2024-02-28
Payment	5000.0	Expense	2024-03-05
Payment	5000.0	Expense	2024-04-16
Fees	5600.0	Expense	2024-04-02
Groceries	6500.0	Expense	2024-03-25
Payment	7000.0	Expense	2024-03-26
Shopping	7500.0	Expense	2024-03-08
Shopping	7800.0	Expense	2024-03-03
Current bill	8000.0	Expense	2024-02-27
Food	8000.0	Expense	2024-03-14
Current bill	8200.0	Expense	2024-03-27
Shopping	8500.0	Expense	2024-04-04
Current bill	8500.0	Expense	2024-04-27
Hotel	9000.0	Expense	2024-04-07
Groceries	11000.0	Expense	2024-03-11
Groceries	12000.0	Expense	2024-02-17

Figure 8-Sorting transaction amount in ascending order



Category ↓	Amount ↓	Type	Date
Salary	180000.0	Income	2024-04-05
Salary	180000.0	Income	2024-03-05
Salary	180000.0	Income	2024-02-05
Gift	100000.0	Income	2024-02-19
Bonus	55000.0	Income	2024-04-10
Interest	40000.0	Income	2024-04-22
Interest	40000.0	Income	2024-03-22
Interest	40000.0	Income	2024-02-22
House rent	35000.0	Expense	2024-02-15
Gift	30000.0	Income	2024-04-20
House rent	30000.0	Expense	2024-04-15
House rent	30000.0	Expense	2024-03-15
Shopping	30000.0	Expense	2024-02-10
Shopping	27000.0	Expense	2024-04-21
Shopping	26000.0	Expense	2024-02-21
Credit card	25000.0	Expense	2024-03-19
Insurance	20000.0	Expense	2024-04-23
Insurance	20000.0	Expense	2024-03-23
Insurance	19000.0	Expense	2024-02-23
Shopping	19000.0	Expense	2024-03-19
Groceries	18000.0	Expense	2024-04-09
Groceries	14000.0	Expense	2024-02-07
Hotel	12500.0	Expense	2024-03-09
Groceries	12000.0	Expense	2024-02-17

Figure 9-Sorting transaction amount in descending order



Category ↓	Amount ↓	Type ↑	Date
Groceries	14000.0	Expense	2024-02-07
Groceries	12000.0	Expense	2024-02-17
Groceries	11000.0	Expense	2024-03-11
Groceries	6500.0	Expense	2024-03-25
Groceries	18000.0	Expense	2024-04-09
Groceries	4500.0	Expense	2024-04-28
Shopping	30000.0	Expense	2024-02-10
Shopping	26000.0	Expense	2024-02-21
Shopping	7800.0	Expense	2024-03-03
Shopping	7500.0	Expense	2024-03-08
Shopping	19000.0	Expense	2024-03-19
Shopping	8500.0	Expense	2024-04-04
Shopping	27000.0	Expense	2024-04-21
House rent	35000.0	Expense	2024-02-15
House rent	30000.0	Expense	2024-03-15
House rent	30000.0	Expense	2024-04-15
Mobile bill	1500.0	Expense	2024-02-18
Mobile bill	1600.0	Expense	2024-03-24
Mobile bill	1500.0	Expense	2024-04-18
Insurance	19000.0	Expense	2024-02-23
Insurance	20000.0	Expense	2024-03-23
Insurance	20000.0	Expense	2024-04-23
Water bill	2500.0	Expense	2024-02-24
Water bill	2300.0	Expense	2024-03-24

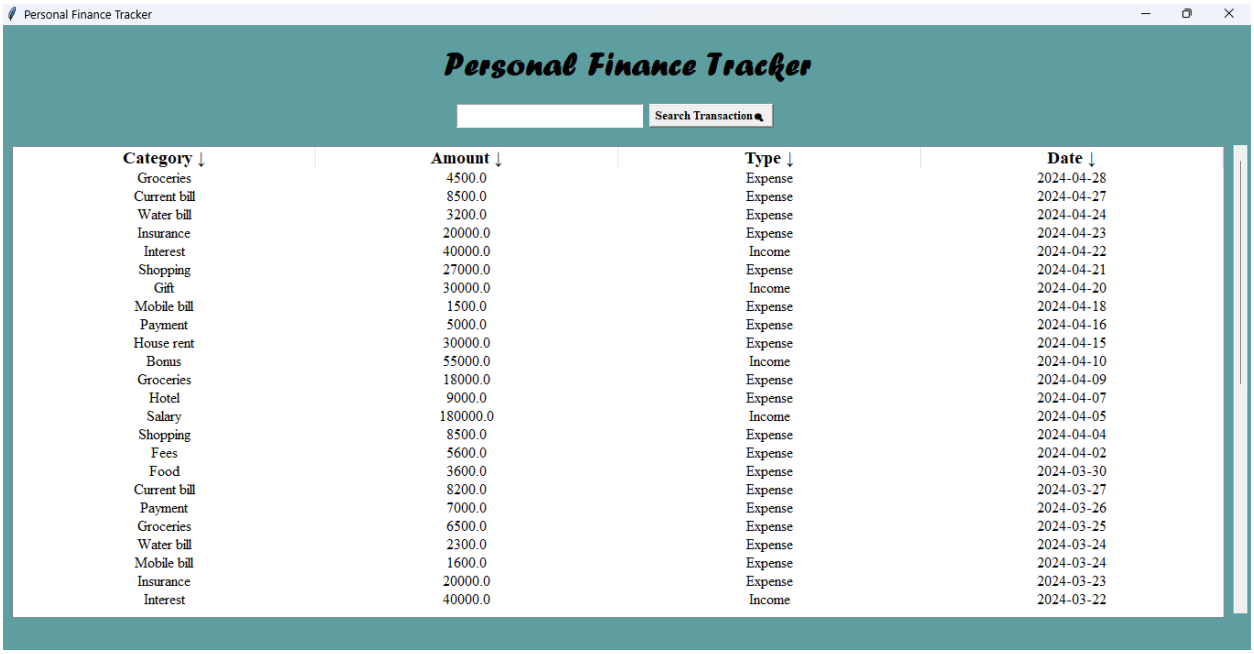
Figure 10-Sorting transaction type in ascending order

Category ↓	Amount ↓	Type ↓	Date
Bonus	55000.0	Income	2024-04-10
Interest	40000.0	Income	2024-04-22
Interest	40000.0	Income	2024-03-22
Interest	40000.0	Income	2024-02-22
Gift	30000.0	Income	2024-04-20
Gift	100000.0	Income	2024-02-19
Salary	180000.0	Income	2024-04-05
Salary	180000.0	Income	2024-03-05
Salary	180000.0	Income	2024-02-05
Credit card	25000.0	Expense	2024-03-19
Food	3600.0	Expense	2024-03-30
Food	8000.0	Expense	2024-03-14
Hotel	9000.0	Expense	2024-04-07
Hotel	12500.0	Expense	2024-03-09
Payment	5000.0	Expense	2024-04-16
Payment	7000.0	Expense	2024-03-26
Payment	5000.0	Expense	2024-03-05
Fees	5600.0	Expense	2024-04-02
Fees	4500.0	Expense	2024-02-28
Current bill	8500.0	Expense	2024-04-27
Current bill	8200.0	Expense	2024-03-27
Current bill	8000.0	Expense	2024-02-27
Water bill	3200.0	Expense	2024-04-24
Water bill	2300.0	Expense	2024-03-24

Figure 11-Sorting transaction type in descending order

Category ↓	Amount ↓	Type ↓	Date ↑
Salary	180000.0	Income	2024-02-05
Groceries	14000.0	Expense	2024-02-07
Shopping	30000.0	Expense	2024-02-10
House rent	35000.0	Expense	2024-02-15
Groceries	12000.0	Expense	2024-02-17
Mobile bill	1500.0	Expense	2024-02-18
Gift	100000.0	Income	2024-02-19
Shopping	26000.0	Expense	2024-02-21
Interest	40000.0	Income	2024-02-22
Insurance	19000.0	Expense	2024-02-23
Water bill	2500.0	Expense	2024-02-24
Current bill	8000.0	Expense	2024-02-27
Fees	4500.0	Expense	2024-02-28
Shopping	7800.0	Expense	2024-03-03
Salary	180000.0	Income	2024-03-05
Payment	5000.0	Expense	2024-03-05
Shopping	7500.0	Expense	2024-03-08
Hotel	12500.0	Expense	2024-03-09
Groceries	11000.0	Expense	2024-03-11
Food	8000.0	Expense	2024-03-14
House rent	30000.0	Expense	2024-03-15
Shopping	19000.0	Expense	2024-03-19
Credit card	25000.0	Expense	2024-03-19
Interest	40000.0	Income	2024-03-22

Figure 12-Sorting transaction date in ascending order



Category ↓	Amount ↓	Type ↓	Date ↓
Groceries	4500.0	Expense	2024-04-28
Current bill	8500.0	Expense	2024-04-27
Water bill	3200.0	Expense	2024-04-24
Insurance	20000.0	Expense	2024-04-23
Interest	40000.0	Income	2024-04-22
Shopping	27000.0	Expense	2024-04-21
Gift	30000.0	Income	2024-04-20
Mobile bill	1500.0	Expense	2024-04-18
Payment	5000.0	Expense	2024-04-16
House rent	30000.0	Expense	2024-04-15
Bonus	55000.0	Income	2024-04-10
Groceries	18000.0	Expense	2024-04-09
Hotel	9000.0	Expense	2024-04-07
Salary	180000.0	Income	2024-04-05
Shopping	8500.0	Expense	2024-04-04
Fees	5600.0	Expense	2024-04-02
Food	3600.0	Expense	2024-03-30
Current bill	8200.0	Expense	2024-03-27
Payment	7000.0	Expense	2024-03-26
Groceries	6500.0	Expense	2024-03-25
Water bill	2300.0	Expense	2024-03-24
Mobile bill	1600.0	Expense	2024-03-24
Insurance	20000.0	Expense	2024-03-23
Interest	40000.0	Income	2024-03-22

Figure 13-Sorting transaction date in decending order

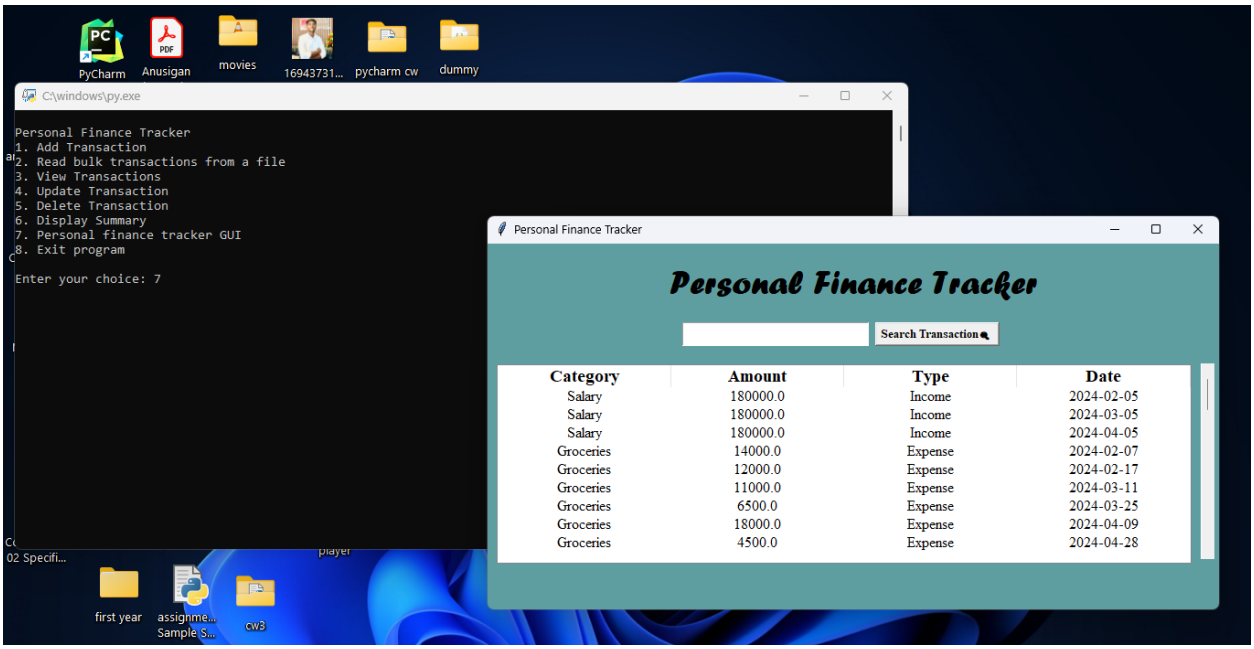


Figure 14-Intergrated with CLI of coursework part B