CST1510 FINAL COURSEWORK B

VENDING MACHINE

Student Details:

Name: Mahek Devang Trivedi

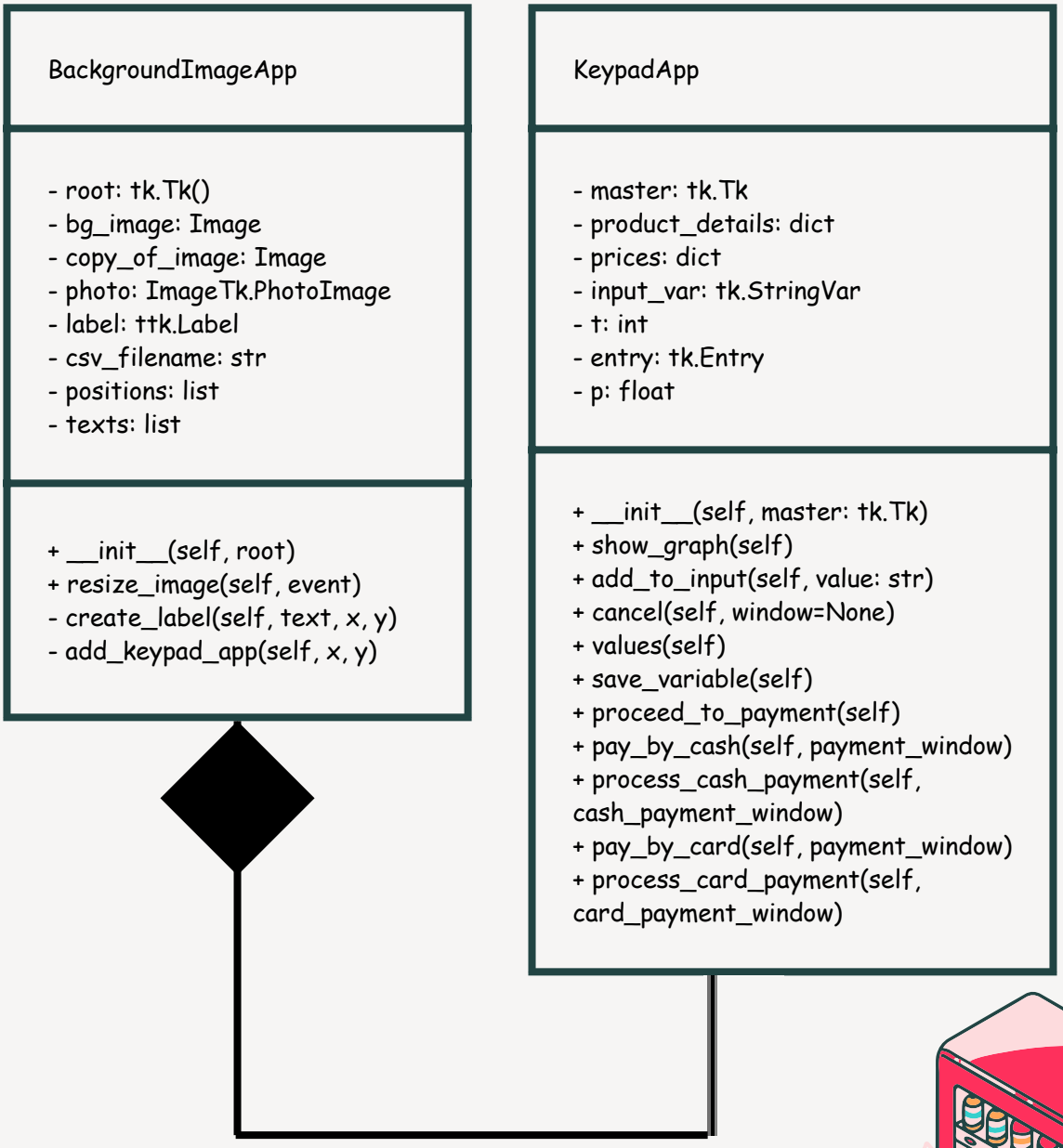
INDEX

INTRODUCTION

This vending machine system is designed to facilitate product purchases through a graphical user interface (GUI) implemented in Python using Tkinter. It allows users to select products using a keypad interface. supports both cash and card payments. and includes features like stock management and transaction logging. The system comprises a client application for user interaction and a server script for handling transactions and stock updates.

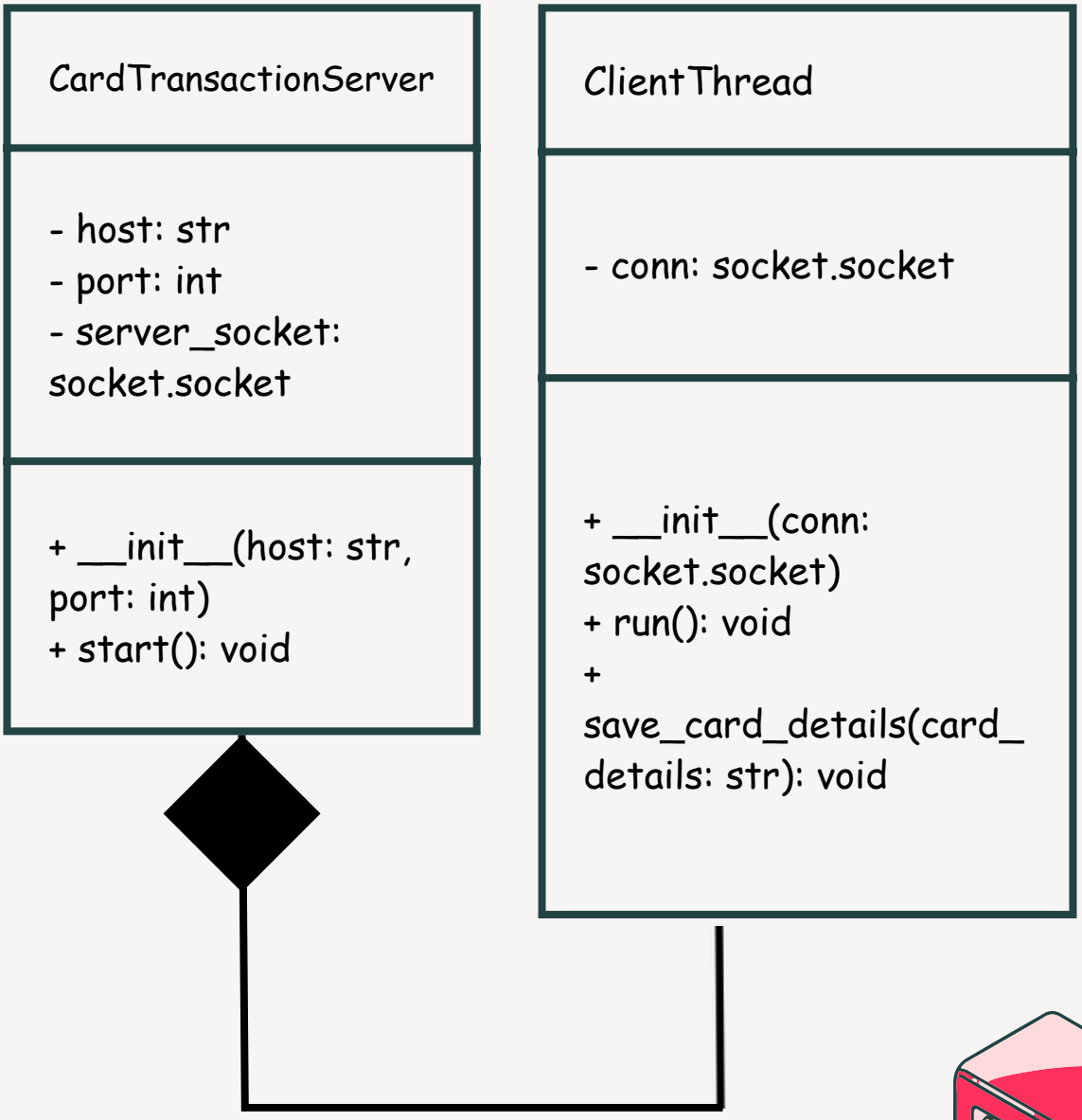
UML DIAGRAM

CLIENT.PY



This represents that BackgroundImageApp has a composition relationship with KeypadApp, indicating that KeypadApp is part of the BackgroundImageApp object.
In the BackgroundImageApp, KeypadApp is composed to handle user input for product selection and payment methods, serving as an integral component within the graphical user interface. The BackgroundImageApp orchestrates the overall application interface, utilizing KeypadApp to manage user interactions related to product selection and payment.

UML DIAGRAM
SERVER.PY



The composition relationship between CardTransactionServer and ClientThread signifies that the lifetime of ClientThread instances is dependent on CardTransactionServer instances. When a CardTransactionServer instance is created, it creates and manages ClientThread instances for handling client connections, and when the server instance is terminated, associated ClientThread instances are also terminated.

IMPLEMENTATION PLAN

1. Understanding the Design: Understand the design provided in the project requirements. Identify the main components, user interactions, and data flow between different parts of the system.
2. Setting Up the Environment: Install the necessary tools and libraries such as Python, Tkinter, PIL (Pillow), and Matplotlib.
3. Creating the GUI: Implement the graphical user interface (GUI) using Tkinter. This involves creating windows, frames, labels, buttons, and entry fields as per the design specifications.
4. Implementing Client-Side Logic: Implement handling button clicks, processing user inputs, displaying messages, and communicating with the server.
5. Server-Side Development: Implement handling client connections, processing requests, and updating product quantities or saving card details as necessary. Make sure it can handle Multi-Client.
6. Testing: Test each component of the system individually to ensure they function correctly. Verify the GUI behaviour, data processing, and network communication.
7. Integration and Debugging: Integrate the client and server components together and debug any issues that arise during integration.
8. Error Handling and Validation: Implement error handling and input validation to handle edge cases and prevent unexpected behaviour. This includes validating user inputs, handling network errors, and displaying appropriate error messages.
9. Optimization and Refinement: Optimize the code for performance and efficiency. Refactor the code if necessary to improve readability, maintainability, and scalability.
10. Documentation and Finalization: Document the implementation details, including code structure, functionality, and usage instructions. Ensure that the codebase is well-documented and organized.

USER MANUAL

1. Starting the Application:
   - Start by execting client.py which opens the main GUI window.
2. Selecting Products:
   - Choose products entering their IDs using the keypad.
   - Press the "ADD" button to add the selected product to your cart.
   - Repeat this process to add multiple products to your cart.
3. Visualising Your Cart:
   - Click the "Show Graph" button to visualize the total price of each product in a graph format.
4. Payment Options:
   - Proceed to payment by clicking the "Proceed To Payment" button. You can see total price and individual details of your product(s).
   - Choose between two payment methods: Cash or Card.
5. Cash Payment:
   - Enter the amount of cash you're paying and click "Submit" to complete the transaction. The change, if any, will be calculated automatically.
6. Card Payment:
   - Provide your card details including Card Holder Name, Card Number, CIV, and Expiry Date (MM/YYYY) in the respective fields.
   - Click "Submit" to process the card payment securely.
7. Payment Confirmation:
   - Upon successful payment, receive a confirmation message.
   - In case of any payment issues, appropriate error messages will be displayed for further action.
8. Exiting the Application:
   - Close the application window or utilize the provided exit button to gracefully exit the vending machine application.

CODE

SERVER.PY

*Pserver.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\Individual\Pserver.py (3.1

File Edit Format Run Options Window Help

```
#Mahek M00979199
#Import necessary libraries
import socket
import csv
import threading
from datetime import datetime


#Define the CardTransactionServer class to handle card transactions
class CardTransactionServer:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind((self.host, self.port))
        self.server_socket.listen(5)  #Listen for up to 5 connections
        print("Server listening on port", self.port)

    def start(self):
        #Continuously accept connections
        while True:
            conn, addr = self.server_socket.accept()
            print("Connected to", addr)
            #Create a new thread for each client connection
            client_thread = ClientThread(conn)
            client_thread.start()

#Define the ClientThread class to handle client connections
class ClientThread(threading.Thread):
    def __init__(self, conn):
        threading.Thread.__init__(self)
        self.conn = conn

    def run(self):
        #Continuously receive and process data from the client
        while True:
            data = self.conn.recv(1024).decode()
            if not data:
                break
```

CODE

SERVER.PY

Pserver.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\Individual\Pserver.py (3.12.2)

File Edit Format Run Options Window Help

```
        #Check if the received data is card details
        if data.startswith("ProdDetails:"):
            card_details = data.replace("ProdDetails:", "")
            self.save_prod_details(card_details)
        else:
            try:
                #Check if the received data is a valid product ID
                num = int(data)
                if 1001 <= num <= 3005:
                    with open('product_quantities.csv', mode='r') as file:
                        reader = csv.DictReader(file)
                        rows = list(reader)
                        for row in rows:
                            if row['ID'] == str(num):
                                if int(row['Quantity']) > 0:
                                    #Update product quantity and send success message
                                    row['Quantity'] = str(int(row['Quantity']) - 1)
                                    with open('product_quantities.csv', mode='w', newline='') as write_file:
                                        writer = csv.DictWriter(write_file, fieldnames=['ID', 'Quantity'])
                                        writer.writeheader()
                                        writer.writerows(rows)
                                        self.conn.sendall("Product purchased successfully".encode())
                                else:
                                    #Send out of stock message
                                    self.conn.sendall("Out of Stock".encode())
                                break
                        else:
                            #Send invalid product ID message
                            self.conn.sendall("Invalid Product ID".encode())
                else:
                    #Send invalid product ID message
                    self.conn.sendall("Invalid Product ID".encode())
            except ValueError:
                #Send invalid input message
                self.conn.sendall("Invalid Input".encode())

    #Close the connection
    self.conn.close()
```

CODE

SERVER.PY

```
    def save_prod_details(self, card_details):
        PROD_DETAILS_HEADERS = ['Transaction_Time', 'Card_Name', 'Card_Number', 'CIV', 'Expiry_Date', 'Total_Price']
        PROD_DETAILS_FILE = "transactions.csv"

        #Save card details along with transaction time
        transaction_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        prod_details_list = [transaction_time] + card_details.split(",")
        with open(PROD_DETAILS_FILE, "a", newline='') as csvfile:
            writer = csv.writer(csvfile)
            if csvfile.tell() == 0:  #Check if file is empty
                writer.writerow(PROD_DETAILS_HEADERS)
            writer.writerow(prod_details_list)


#Define the main function to start the server
def main():
    # Check if CSV file exists, if not, create it
    csv_filename = 'product_quantities.csv'
    if not os.path.exists(csv_filename):
        with open(csv_filename, mode='w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['ID', 'Quantity'])
            ID = ['1001', '1002', '1003', '1004', '1005', '2001', '2002', '2003', '2004', '2005', '3001', '3002', '3003', '3004', '3005']
            for i in ID:
                writer.writerow([i, 10])
    host = '127.0.0.1'
    port = 12345
    card_transaction_server = CardTransactionServer(host, port)
    card_transaction_server.start()

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

CODE

CLIENT.PY

*PClient.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\CST1510_Project_M00979199\PCl

File Edit Format Run Options Window Help

```
#Mahek M00979199
import socket  # For socket communication
import tkinter as tk  # For GUI
from tkinter import ttk, messagebox  # Additional GUI components
from PIL import ImageTk, Image  # For image manipulation
import csv  # For CSV file operations
import matplotlib.pyplot as plt  # For plotting graphs
import os  # For file operations

class BackgroundImageApp:
    def __init__(self, root):
        # Initialize the main application window
        self.root = root
        self.root.title("Vending machine")
        self.root.geometry("1050x590")
        self.root.resizable(False, False)
        # Load background image
        self.bg_image = Image.open("DC.png")
        self.copy_of_image = self.bg_image.copy()
        self.photo = ImageTk.PhotoImage(self.bg_image)
        self.label = ttk.Label(root, image=self.photo)
        self.label.bind('<Configure>', self.resize_image)
        self.label.pack(fill=tk.BOTH, expand=tk.YES)
        # Adding the labels for products
        positions = [(68, 176), (181, 176), (303, 176), (414, 176), (542, 176),
                     (68, 342), (181, 342), (303, 342), (414, 342), (542, 342),
                     (68, 515), (181, 515), (303, 515), (414, 515), (542, 515)]
        texts = ["1001-2$", "1002-1.5$", "1003-3$", "1004-2.5$", "1005-1$",
                 "2001-3$", "2002-4.5$", "2003-6$", "2004-3.5$", "2005-4$",
                 "3001-2$", "3002-3.5$", "3003-5$", "3004-1.5$", "3005-2$"]
        for text, (x, y) in zip(texts, positions):
            self.create_label(text, x, y)
        # Adding KeypadApp at position (710, 95)
        self.add_keypad_app(710, 95)
        messagebox.showinfo("Welcome", "Welcome to the Vending Machine!")
    # Method to create label for products
    def create_label(self, text, x, y):
        label = tk.Label(self.root, text=text, font=("Comic Sans MS", 12, "bold"), fg="white", bg="#f6b2b8")
        label.place(x=x, y=y)
```

*PClient.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\CST1510_Project_M00979199\PClient.py (3.12.

File Edit Format Run Options Window Help

```
        return label
    # Method to add KeypadApp to the main application
    def add_keypad_app(self, x, y):
        keypad_frame = tk.Frame(self.root, bg="#f6b2b8")
        keypad_frame.place(x=x, y=y)
        keypad_app = KeypadApp(keypad_frame)
    # Method to resize the background image according to window size
    def resize_image(self, event):
        new_width = event.width
        new_height = event.height
        image = self.copy_of_image.resize((new_width, new_height))
        photo = ImageTk.PhotoImage(image)
        self.label.config(image=photo)
        self.label.image = photo
```

CODE

CLIENT.PY

PClient.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\CST1510_Project_M00979199\PClient.py (3.12.2)

File Edit Format Run Options Window Help

```
class KeypadApp:
    def __init__(self, master):
        # Initialize the keypad application
        self.master = master
        self.product_details = {}
        self.prices = {}
        self.input_var = tk.StringVar()
        self.input_var.set("")
        self.t=0
        self.entry = tk.Entry(master, textvariable=self.input_var, justify='right', font=('Arial', 14))
        self.entry.grid(row=0, column=0, columnspan=3, padx=10, pady=10, sticky='ew')
        self.p = 0
        buttons = [
            '7', '8', '9',
            '4', '5', '6',
            '1', '2', '3',
            '0', '.', 'C'
        ]
        row = 1
        col = 0
        for button in buttons:
            tk.Button(master, text=button, width=5, height=2, command=lambda b=button: self.add_to_input(b)).grid(row=row, column=col, padx=5, pady=5)
            col += 1
            if col > 2:
                col = 0
                row += 1
        tk.Button(master, text="Show Graph", command=self.show_graph).grid(row=5, column=0, columnspan=3, padx=5, pady=5)
        tk.Button(master, text="ADD", width=5, height=2, command=self.save_variable).grid(row=4, column=0, columnspan=3, padx=5, pady=5)
        tk.Button(master, text="Proceed To Payment", width=20, height=2, command=self.proceed_to_payment).grid(row=7, column=0, columnspan=5, padx=5, pady=5)
    def show_graph(self):
        # Calculate total price for each product
        total_prices = {}
        for product_id, quantity in self.product_details.items():
            total_prices[product_id] = self.prices[product_id] * quantity
        # Plotting the graph
        plt.figure(figsize=(8, 6))
        plt.bar(total_prices.keys(), total_prices.values(), color='skyblue')
        plt.xlabel('Product ID')
        plt.ylabel('Total Price ($)')
```

PClient.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\CST1510_Project_M00979199\PClient.py (3.12.2)

File Edit Format Run Options Window Help

```
        plt.title('Total Price of Each Product')
        plt.xticks(rotation=45)
        plt.tight_layout()
        # Displaying the graph
        plt.show()
    # Method to add input to the entry field
    def add_to_input(self, value):
        current_input = self.input_var.get()
        if value == 'C':
            self.input_var.set("")
        else:
            self.input_var.set(current_input + value)
    #Method to print message when payment is cancelled
    def cancel(self, window=None):
        if window:
            window.destroy()
            messagebox.showinfo("Payment Canceled", "Sorry, we could not provide you with what you would like today. We hope to be seeing you again soon. Wish you a Good day!")
    #Method to communicate with server for card details
    def values(self):
        card_name = self.card_name_var.get()
        card_number = self.card_number_var.get()
        expiry_month = self.expiry_month_var.get()
        expiry_year = self.expiry_year_var.get()
        prod_details = f"{card_name},{card_number},civ:***,{expiry_month}/{expiry_year},{self.p}"
        communicate_with_server("ProdDetails:" + prod_details)
    #Calculate total price
    def save_variable(self):
        num = self.input_var.get()
        response=communicate_with_server(num)
        self.prices = {
        '1001': 2, '1002': 1.5, '1003': 3, '1004': 2.5, '1005': 1,
        '2001': 3, '2002': 4.5, '2003': 6, '2004': 3.5, '2005': 4,
        '3001': 2, '3002': 3.5, '3003': 5, '3004': 1.5, '3005': 2}
        ID=['1001', '1002', '1003', '1004', '1005', '2001', '2002', '2003', '2004', '2005', '3001', '3002', '3003', '3004', '3005']
        Price=[2,1.5,3,2.5,1,3,4.5,6,3.5,4,2,3.5,5,1.5,2]
        try:
            a = ID.index(num)
            if response!= "Out of Stock":
                self.p += Price[a]
```

CODE

CLIENT.PY

```
*PClient.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\CST1510_Project_M00979199\PClient.py (3.12.2)*
File  Edit  Format  Run  Options  Window  Help
            if num in self.product_details:
                self.product_details[num] += 1
            else:
                self.product_details[num] = 1
        except ValueError:
            messagebox.showerror("Error", "Invalid product ID!")
    def proceed_to_payment(self):
        # Create the payment window
        payment_window = tk.Toplevel(self.master)
        payment_window.title("Payment Method")
        payment_window.geometry("424x600")
        # Load the background image and resize it
        payment_bg = tk.PhotoImage(file="Paymentbg.png")
        resized_bg = payment_bg.subsample(1, 1)  # Adjust the subsample values as needed
        # Create a label to display the background image
        bg_label = tk.Label(payment_window, image=resized_bg)
        bg_label.place(x=0, y=0, relwidth=1, relheight=1)
        bg_label.image = resized_bg  # Keep a reference to avoid garbage collection
        # Calculate total price and create labels for amount to pay and buttons for payment methods
        total_price = f"Total Price: ${self.p:.2f}"
        tk.Label(payment_window, text=total_price, font=("Comic Sans MS", 15), fg="#377B4C", bg="#FFF6EA").place(x=132, y=215)
        # Add a section to display product details
        product_details_label = tk.Label(payment_window, text="Product Details", font=("Comic Sans MS", 12), fg="#377B4C", bg="#FFF6EA")
        product_details_label.place(x=155, y=250)
        # Display each product's details
        for i, (product_id, quantity) in enumerate(self.product_details.items(), start=1):
            product_info = f"{product_id} - {quantity} - ${self.prices[product_id] * quantity:.2f}"
            tk.Label(payment_window, text=product_info, font=("Comic Sans MS", 10), fg="#377B4C", bg="#FFF6EA").place(x=155, y=280 + (i - 1) * 30)
        # Add payment buttons
        tk.Button(payment_window, text="Pay by Cash", command=lambda: self.pay_by_cash(payment_window), font=("Comic Sans MS", 11), fg="#377B4C", bg="#FFF6EA").place(x=170, y=405)
        tk.Button(payment_window, text="Pay by Card", command=lambda: self.pay_by_card(payment_window), font=("Comic Sans MS", 11), fg="#377B4C", bg="#FFF6EA").place(x=170, y=455)
    def pay_by_cash(self, payment_window):
        payment_window.destroy()
        cash_payment_window = tk.Toplevel(self.master)
        cash_payment_window.title("Cash Payment")
        cash_payment_window.geometry("500x203")
        # Load the background image and resize it to 500x203
        bg_image = tk.PhotoImage(file="Paybg.png")
        resizedbg = bg_image.subsample(1, 1)# Adjust the subsample factor as needed
```

```
*PClient.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\CST1510_Project_M00979199\PClient.py (3.12.2)*
File  Edit  Format  Run  Options  Window  Help
        # Create a label to hold the background image
        bg_label = tk.Label(cash_payment_window, image=resizedbg)
        bg_label.place(x=0, y=0, relwidth=1, relheight=1)
        bg_label.image = resizedbg
        tk.Label(cash_payment_window, text=f"Total amount to Pay: {self.p}$", font=("Comic Sans MS", 15), fg="#2C4975", bg="#CFEADB").place(x=225,y=32)
        tk.Label(cash_payment_window, text="Enter the amount:", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").place(x=260,y=70)
        self.amount_var = tk.StringVar()
        tk.Entry(cash_payment_window, textvariable=self.amount_var).place(x=265,y=100)
        tk.Button(cash_payment_window, text="Submit", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB", command=lambda: self.process_cash_payment(cash_payment_window)).place(x=252,y=130)
        tk.Button(cash_payment_window, text="Cancel", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB", command=lambda: self.cancel(cash_payment_window)).place(x=345,y=130)
    #Processing cash payment, asking to pay if insufficient/returning change
    def process_cash_payment(self,cash_payment_window):
        self.amount_to_pay=self.p
        try:
            cash_amount = float(self.amount_var.get())
            self.t+=cash_amount
            if self.t < self.amount_to_pay:
                remaining_amount = self.amount_to_pay - self.t
                messagebox.showerror("Error", f"Insufficient amount! Please pay ${remaining_amount:.2f} more.")
            elif self.t == self.amount_to_pay:
                messagebox.showinfo("Success", "Payment successful! Thank you! Goodbye!")
                cash_payment_window.destroy()
                root.destroy()
                prod_details = f"-Cash-","-NA-","-NA-","-NA-","{self.p}"
                communicate_with_server("ProdDetails:" + prod_details)
            else:
                change = self.t - self.amount_to_pay
                messagebox.showinfo("Success", f"Payment successful! Your change: ${change:.2f} Thank you! Goodbye!")
                cash_payment_window.destroy()
                root.destroy
                prod_details = f"-Cash-","-NA-","-NA-","-NA-","{self.p}"
                communicate_with_server("ProdDetails:" + prod_details)
        except ValueError:
            messagebox.showerror("Error", "Please enter a valid amount!")
    def pay_by_card(self, payment_window):
        payment_window.destroy()
        card_payment_window = tk.Toplevel(self.master)
        card_payment_window.title("Card Payment")
        card_payment_window.geometry("300x424")
```

CODE

CLIENT.PY

PClient.py - C:\Users\Admin\AppData\Local\Programs\Python\Python312\CST1510_Project_M00979199\PClient.py (3.12.2)

File Edit Format Run Options Window Help

```
    # Load the background image and resize it to 500x203
    bgimage = tk.PhotoImage(file="Cardbg.png")
    resizedBG = bgimage.subsample(1, 1)# Adjust the subsample factor as needed
    # Create a label to hold the background image
    bg_label = tk.Label(card_payment_window, image=resizedBG)
    bg_label.place(x=0, y=0, relwidth=1, relheight=1)
    bg_label.image = resizedBG
    tk.Label(card_payment_window, text=f"Total amount to Pay: {self.p}$", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").pack(pady=10)
    tk.Label(card_payment_window, text="Enter Card Details:", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").pack()
    tk.Label(card_payment_window, text="Card Holder Name:", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").pack()
    self.card_name_var = tk.StringVar()
    tk.Entry(card_payment_window, textvariable=self.card_name_var).pack()
    tk.Label(card_payment_window, text="Card Number:", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").pack()
    self.card_number_var = tk.StringVar()
    tk.Entry(card_payment_window, textvariable=self.card_number_var).pack()
    tk.Label(card_payment_window, text="CIV:", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").pack()
    self.civ_var = tk.StringVar()
    tk.Entry(card_payment_window, textvariable=self.civ_var, show="*").pack()
    tk.Label(card_payment_window, text="Expiry (MM/YYYY):", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").pack()
    expiry_frame = tk.Frame(card_payment_window)
    expiry_frame.pack()
    self.expiry_month_var = tk.StringVar()
    tk.Entry(expiry_frame, textvariable=self.expiry_month_var, width=2).pack(side=tk.LEFT)
    tk.Label(expiry_frame, text="/", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB").pack(side=tk.LEFT)
    self.expiry_year_var = tk.StringVar()
    tk.Entry(expiry_frame, textvariable=self.expiry_year_var, width=4).pack(side=tk.LEFT)
    tk.Button(card_payment_window, text="Submit", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB", command=lambda:self.process_card_payment(card_payment_window)).pack(pady=7)
    tk.Button(card_payment_window, text="Cancel", font=("Comic Sans MS", 12), fg="#2C4975", bg="#CFEADB", command=lambda: self.cancel(card_payment_window)).pack(pady=7)
  #Process card payment,  handle errors
  def process_card_payment(self,card_payment_window):
    card_name = self.card_name_var.get()
    card_number = self.card_number_var.get()
    civ = self.civ_var.get()
    expiry_month = self.expiry_month_var.get()
    expiry_year = self.expiry_year_var.get()
    if not (card_name and card_number and civ and expiry_month and expiry_year):
      messagebox.showerror("Error", "Please fill all fields.")
    elif len(card_number) != 16:
      messagebox.showerror("Error", "Invalid Card Number.")
```

```
      elif len(civ) != 3:
        messagebox.showerror("Error", "Invalid CIV.")
      elif not (expiry_month.isdigit() and expiry_year.isdigit()):
        messagebox.showerror("Error", "Invalid Expiry Date.")
      elif int(expiry_year) < 2023 or not (1 <= int(expiry_month) <= 12):
        messagebox.showerror("Error", "Invalid Expiry Date.")
      else:
        messagebox.showinfo("Payment Successful", "Payment successful. Thank you! Goodbye!")
        card_payment_window.destroy()
        root.destroy()
        self.values()
#Function to communicate with server
def communicate_with_server(product_id):
  host = '127.0.0.1'
  port = 12345
  try:
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))
    client_socket.sendall(product_id.encode())
    response = client_socket.recv(1024).decode()
    messagebox.showinfo("Server Response", response)
    return response
  except Exception as e:
    messagebox.showerror("Error", str(e))
  finally:
    client_socket.close()
root = tk.Tk()
app = BackgroundImageApp(root)
root.mainloop()
```

CODE EXPLANATION

CLIENT.PY

a. Imports:
   i. Socket for server communication, tkinter for GUI, PIL for image manipulation, csv for CSV operations, matplotlib.pyplot for plotting, os for file operations.

b. BackgroundImageApp Class:
   i. Initializes the main application window with a background image loaded from "DC.png".
   ii. Adds labels for product information on the GUI, including product IDs and prices.
   iii. Initializes a keypad application for user input and displays a welcome message.

c. KeypadApp Class:
   i. Initializes the keypad application within the main GUI, allowing users to input values.
   ii. Provides methods to:
      1. Add input to the entry field, and display a graph.
      2. Save user input as a variable and calculate the total price of selected products.
      3. Proceed to payment, by choosing cash or card payment methods.
      4. Process cash or card payments, handle errors, and communicate payment details with a server.

d. Communication with Server:
   i. Defines the communicate_with_server function to establish a socket connection with a server, send product details, and receive responses.

a. Main Section:
   i. Creates the main Tkinter window, initializes the BackgroundImageApp instance, and starts the GUI event loop with root.mainloop().

CODE EXPLANATION

SERVER.PY

a. Import necessary libraries: Import required libraries for socket programming, CSV file handling, threading, and date/time operations respectively.
Such as:
i. socket
ii. csv
iii. threading
iv. datetime

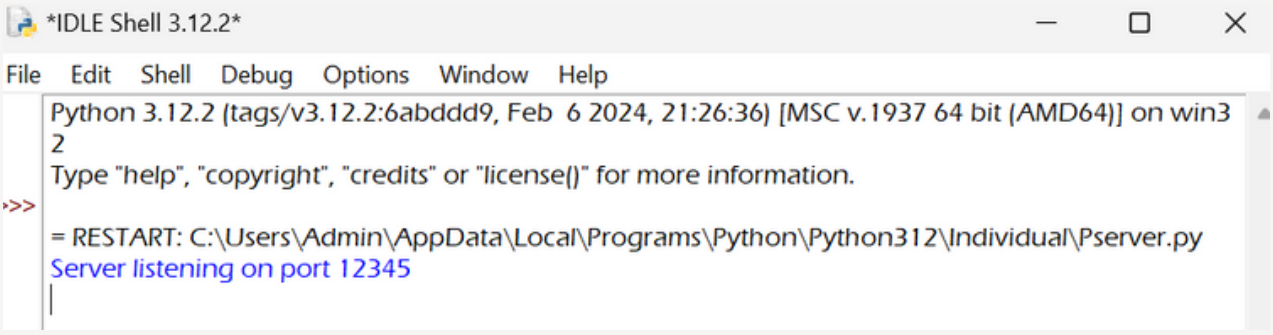
b. Define the CardTransactionServer class:
This class is responsible for handling card transactions. It initializes a server socket, binds it to a host and port, and listens for incoming connections. When a connection is accepted, it creates a new thread (ClientThread) to handle the client connection.

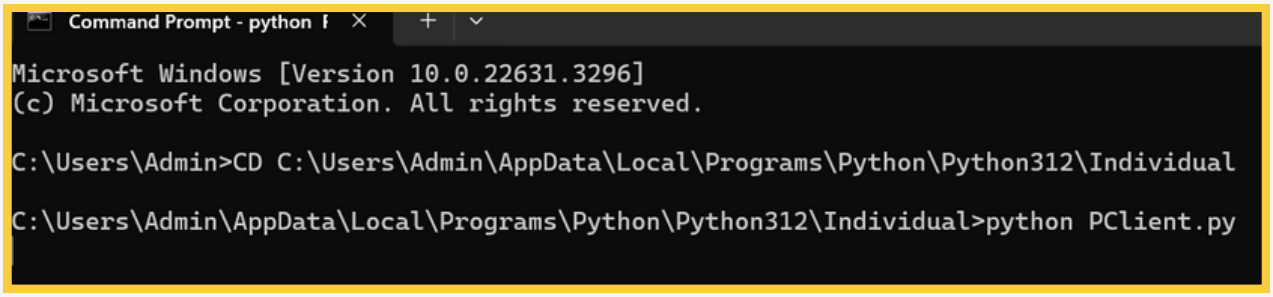
c. Define the ClientThread class:
This class, which extends threading.Thread handles client connections. It continuously receives data from the client, processes it, and responds accordingly. If the received data contains product details, it saves them to a CSV file. If it's a product ID, it checks its validity, updates the product quantity if available, and sends appropriate messages back to the client.

d. Define the main() function:
This function sets the host and port, creates an instance of CardTransactionServer, and starts the server.
Checks if a CSV file named "product_quantities.csv" exists. If not, it creates one with default product IDs and quantities.

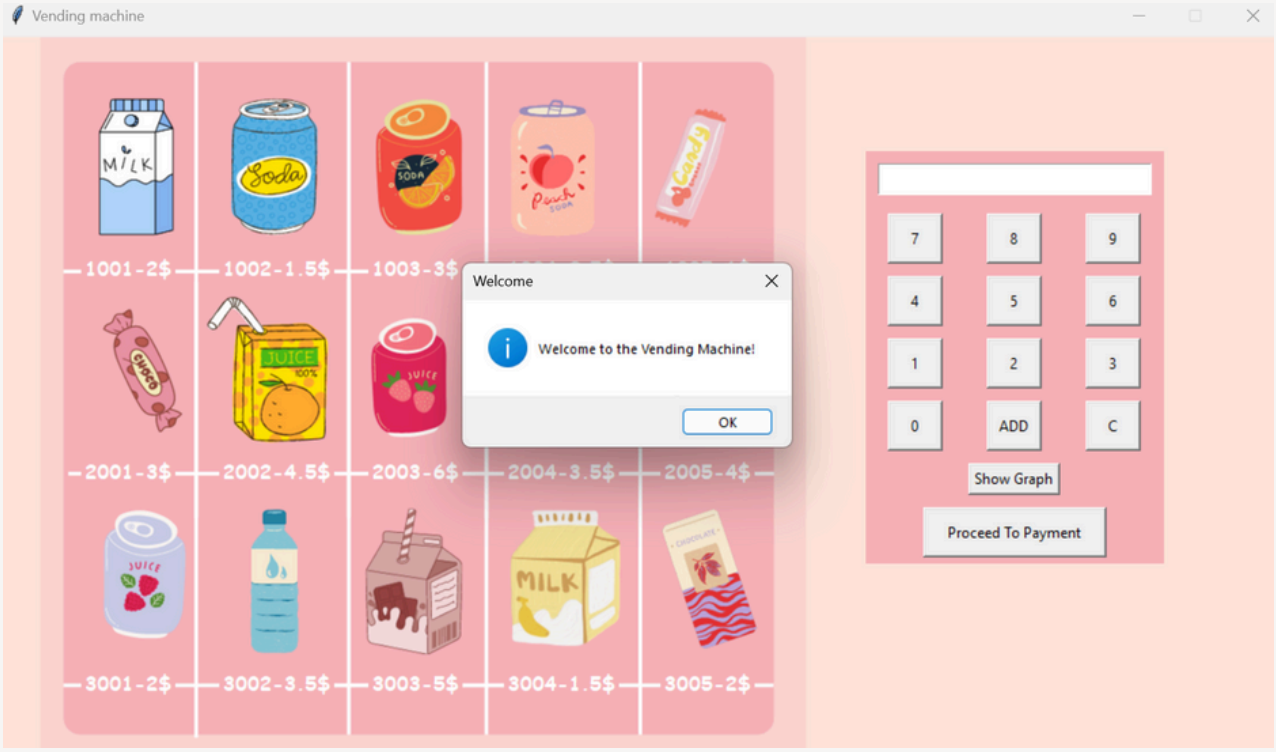
e. Start the server:
The script checks if it's being run directly (if __name__ == "__main__":) and if so, it calls the main() function to start the server.

OUTPUT





The welcome message:



OUTPUT

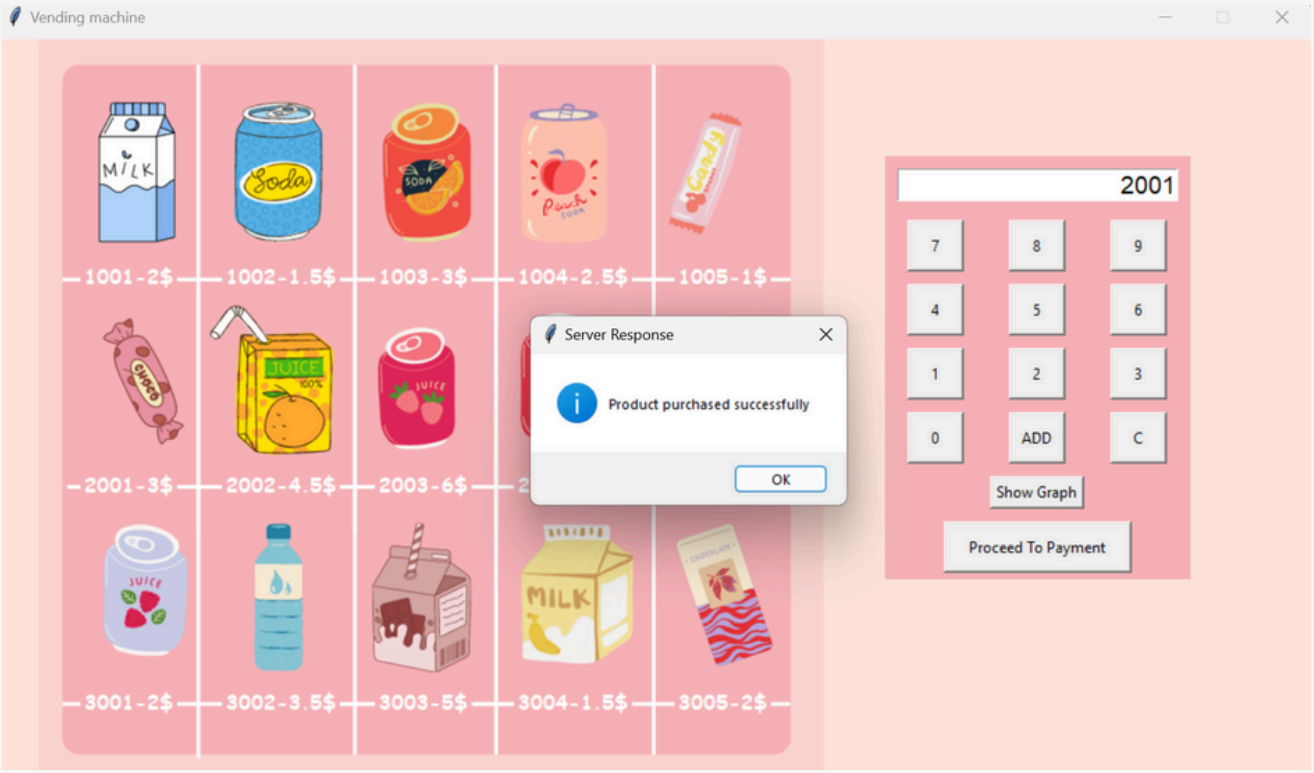
POSSIBLE DATA LOSS Some features might be los

P14

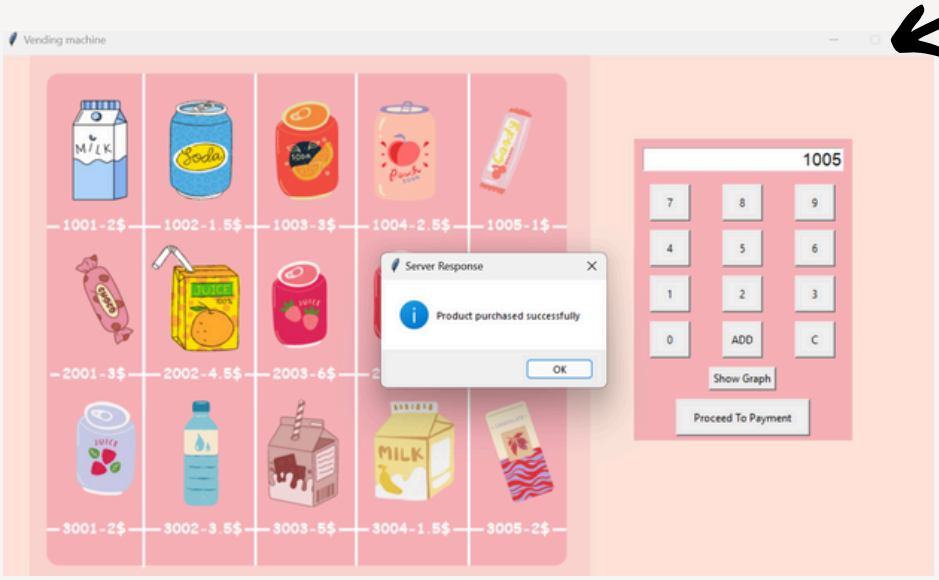
	A	B	C	D
1	ID	Quantity		
2	1001	10		
3	1002	10		
4	1003	10		
5	1004	10		
6	1005	10		
7	2001	10		
8	2002	10		
9	2003	10		
10	2004	10		
11	2005	10		
12	3001	10		
13	3002	10		
14	3003	10		
15	3004	10		
16	3005	10		

The code creates a csv file to store quantities of each product:

We start by purchasing our first product by entering the ID using the keypad. We then click the button "ADD" to add the item to the card. Right after adding a message box tell's us the product has been purchased successfully. Here, we have purchased the item with the ID 2001.

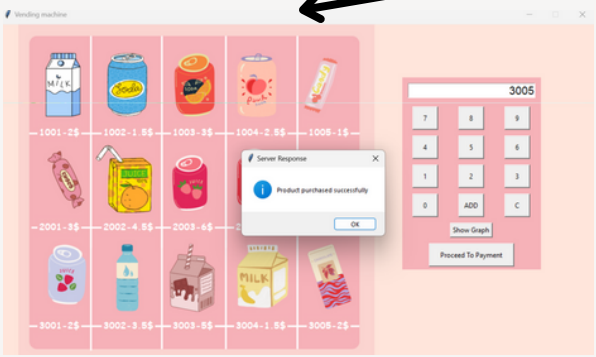


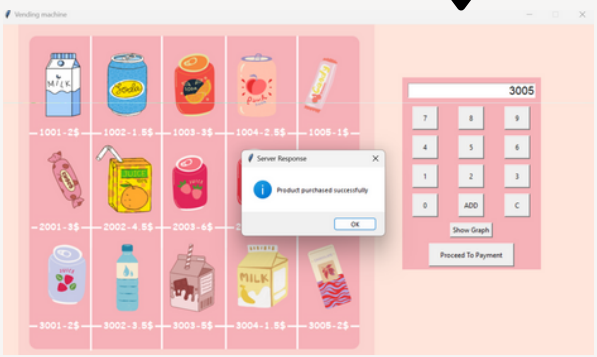
OUTPUT



Now, we have purchased the item with the ID 1005

Next we purchase the product with the ID 3005 twice

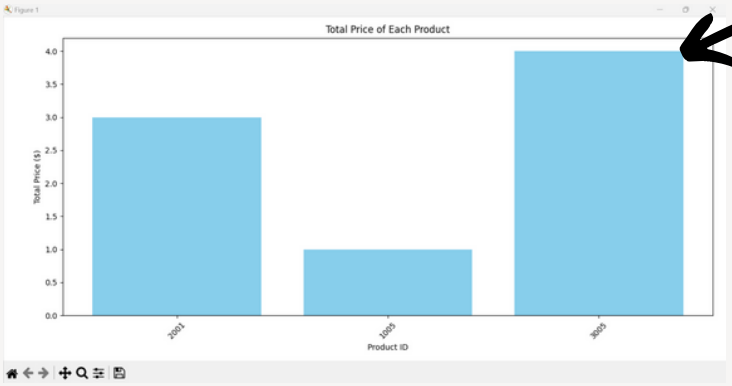




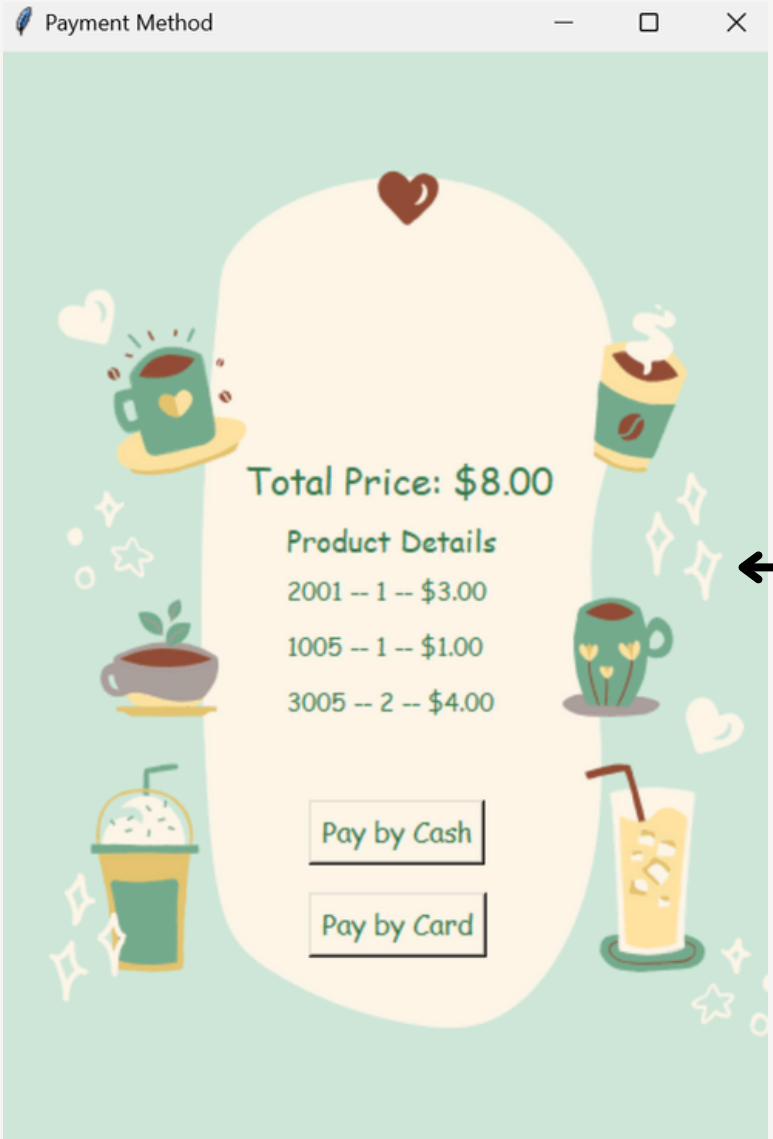
	A	B	C	D
1	ID	Quantity		
2	1001	10		
3	1002	10		
4	1003	10		
5	1004	10		
6	1005	9		
7	2001	9		
8	2002	10		
9	2003	10		
10	2004	10		
11	2005	10		
12	3001	10		
13	3002	10		
14	3003	10		
15	3004	10		
16	3005	8		

The quantities have then been updated in the csv file by the server. The quantity of products with ID 1005 and 2001 have been changed to 9 while the quantity of the product with ID 3005 has been changed to 8.

OUTPUT



When you click the button "Show Graph" given on the keypad, it shows you a visual representation of your purchase summary in the form of a graph



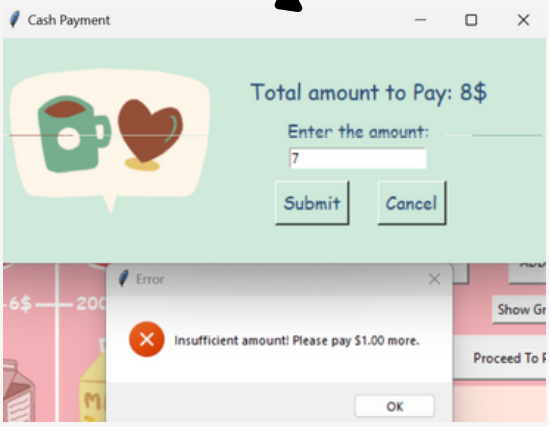
After purchasing all your products you can now click "Proceed To Payment."
That will transfer you to this page. You can see the total cost and product details of your individual products such as Product ID of the product you have purchased, the quantity and the price.

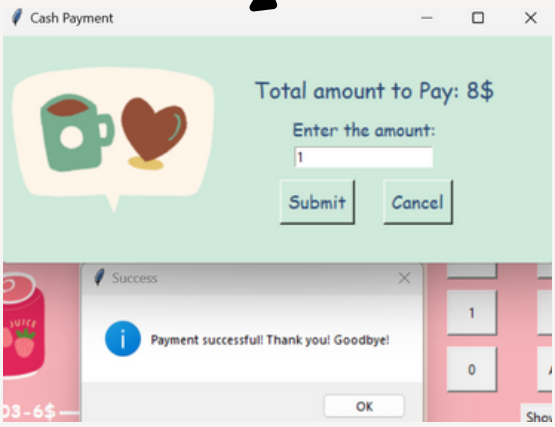
The page will ask you to choose if you want to pay by cash or by card.

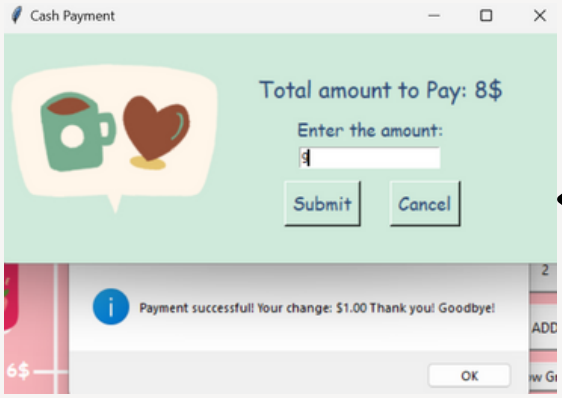
OUTPUT

Pay By Cash

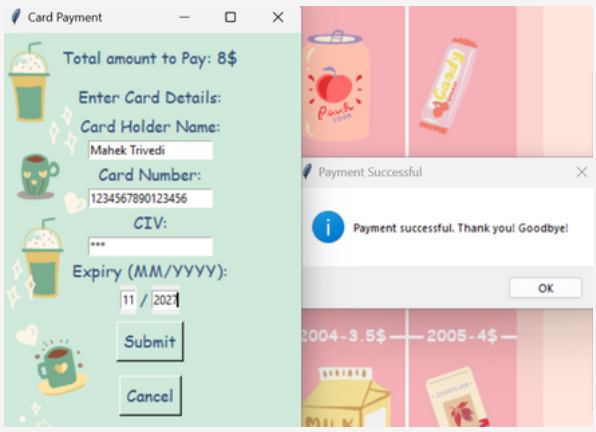
If you pay less than expected:







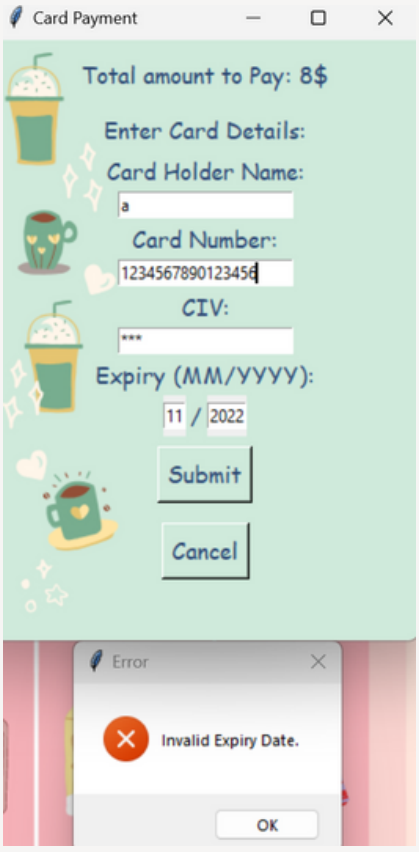
If you pay more than expected, your change will be automatically calculated and returned.

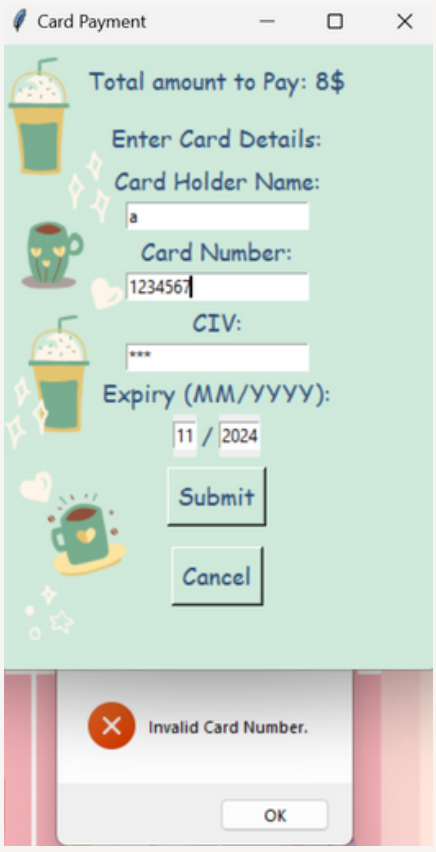


Pay By Card

If you choose to pay by card, you will be asked to enter information such as Card Holder Name, Card Number, CIV and the Expiry in terms of month and year.

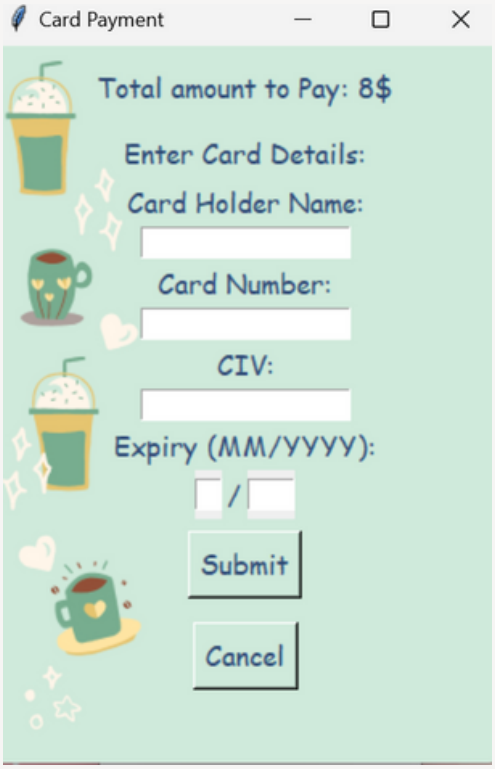
OUTPUT

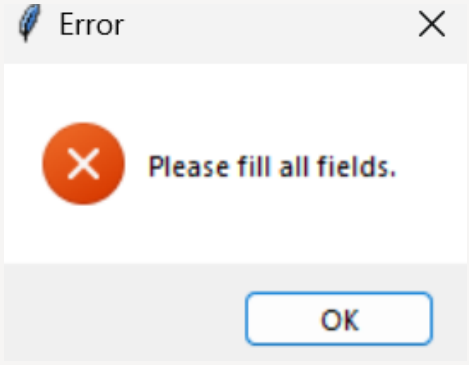




You will be asked to enter you card details again in case you-

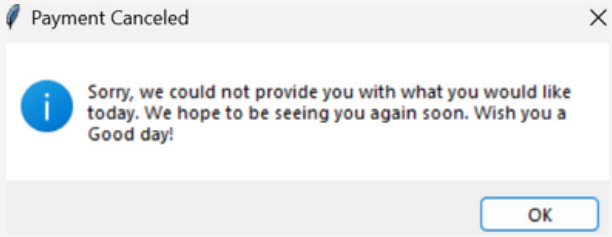
1. Leave any entry empty
2. Enter a Card Number that is not 16 digits long
3. Enter a date that has already gone by as the expiry



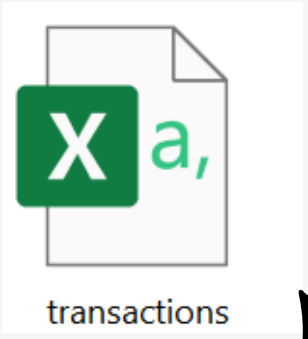


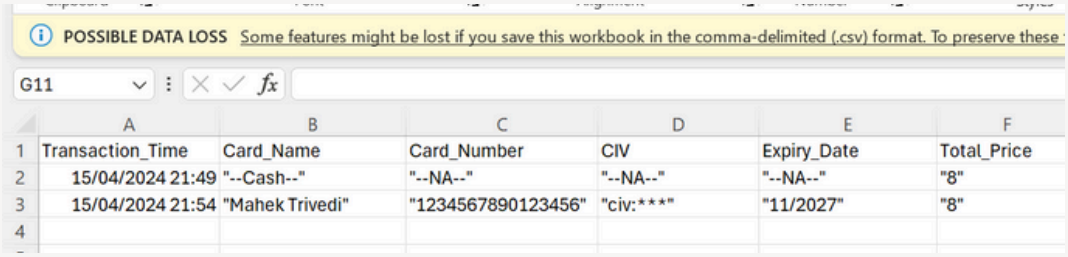
OUTPUT

If you cancel the payment,
the given message will be displayed



The transaction details for both,
card payment and cash payment are stored
in a csv file called transactions.csv



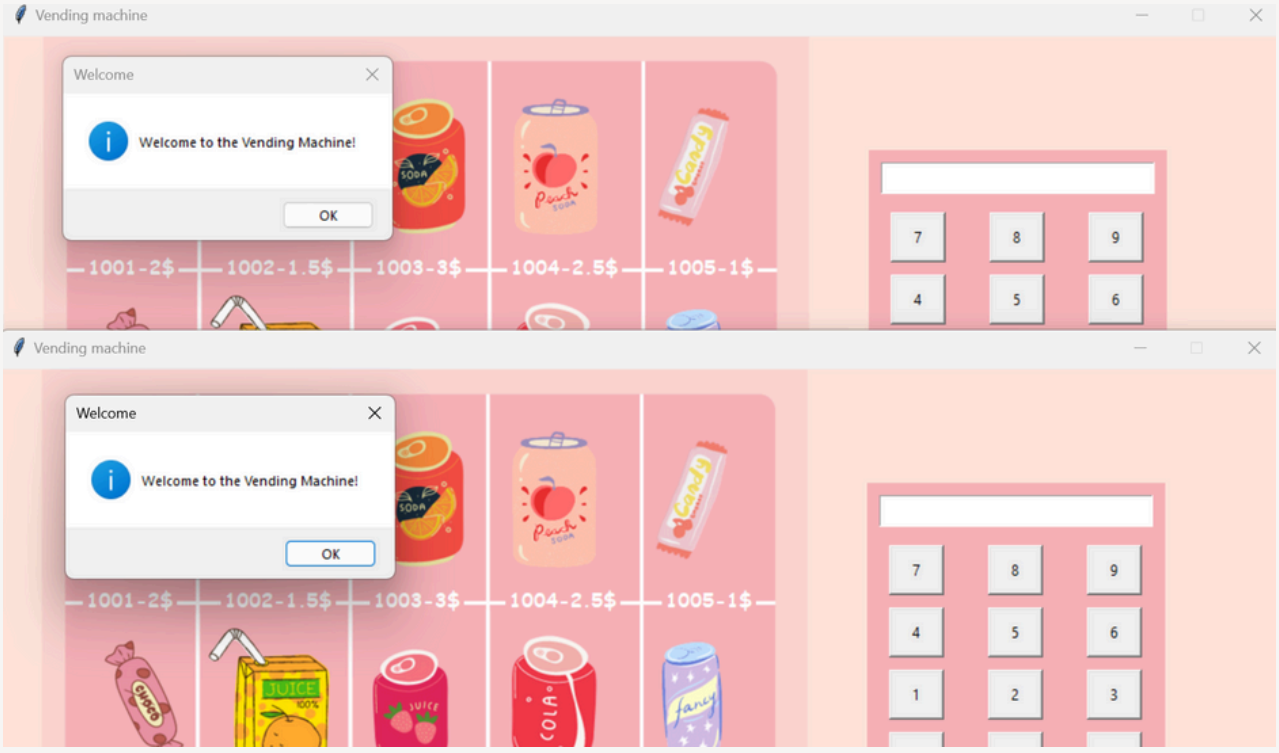


POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these

G11

	A	B	C	D	E	F
1	Transaction_Time	Card_Name	Card_Number	CIV	Expiry_Date	Total_Price
2	15/04/2024 21:49	"--Cash--"	"--NA--"	"--NA--"	"--NA--"	"8"
3	15/04/2024 21:54	"Mahek Trivedi"	"1234567890123456"	"civ:***"	"11/2027"	"8"
4						

The vending machine is multiclient. The server can
handle up to 5 clients at a time:



FUTURE PLANS

1. Modifying the csv file Product_Quantities to show various kinds of reports including Purchases, Sales and Balance Amount.
2. Modifying the csv file transactions to also include the quantity of products next to the price column.
3. User Accounts and Preferences: Introducing user accounts allows individuals to create profiles within the vending machine system. With these profiles, users can save their favorite products, track their purchase history, and receive personalized recommendations. User accounts enhance engagement, loyalty, and provide valuable data for targeted marketing.
4. Mobile Payment Integration: By integrating mobile payment options like NFC or QR code scanning.
5. Product Recommendations: Implementing product recommendation algorithms enables the vending machine to analyze user behavior and preferences. Based on this analysis, the machine can suggest relevant products to individual users, increasing the likelihood of purchase and enhancing customer satisfaction.
6. Customer Feedback Mechanism: Introducing a feedback mechanism allows users to provide their opinions and suggestions directly to the vending machine operator. Feedback forms or interactive surveys gather valuable insights into user preferences, satisfaction levels, and areas for improvement. Implementing customer feedback mechanisms demonstrates a commitment to customer-centricity and continuous improvement.

BIBLIOGRAPHY

1. https://stackoverflow.com/
2. https://docs.python.org/3/library/tkinter.html
3. https://www.geeksforgeeks.org/
4. https://pillow.readthedocs.io/en/stable/
5. https://realpython.com/
6. https://forums.raspberrypi.com/viewtopic.php?t=308771

REFLECTION

Building the multiclient vending machine system was a comprehensive endeavour that integrated frontend GUI with backend server-client communication. Key aspects included:

1. Integration of Frontend and Backend: The project seamlessly combined Tkinter GUI development with socket-based server-client communication, ensuring a user-friendly interface while maintaining robust backend functionality.
2. Concurrency Handling: Multithreading in the server enabled simultaneous handling of multiple client connections, optimizing system responsiveness and resource utilization.
3. Data Persistence and Logging: CSV operations managed product quantities and transaction logging, enhancing system reliability and accountability.
4. Error Handling and User Feedback: Effective error handling and message prompts guided users through the purchasing process, ensuring a smooth user experience.
5. Security Considerations: While functionality was prioritized, future iterations could focus on enhancing security measures, especially regarding card payment processing.
6. Scalability and Extensibility: The modular design allowed for scalability and easy addition of new features, ensuring the system's adaptability to evolving requirements.

THANK YOU