APRIL 8, 2020

CST 1510 Programming for Data Communication and Networks

Final Coursework- Python Vending Machine

GUI and Socket Programming

STUDENT ID:
M00796614

Contents

Introduction

This final coursework had for aim the designing of a Vending Machine with the implementation of the Python programming tools we learnt in this CST 1510 module throughout the academic year.

In order to achieve an intuitive interactive user interface, the Tkinter module was implemented while the Socket low-level networking interface was used to ensure the building of a Client/Server-side software system. To further provide visual representation within the graphical user interface (GUI), Matplotlib alongside Pandas libraries were used to create a bar chart.

The Vending Machine designed in this coursework is entitled “Coffee Beans”. It sells a variety of hot and cold coffee. To begin with this project, a coffee.csv file was created which contained all the vital information of the products. The data stored in the .csv file is as follows:

- → Product ID
- → Product Name
- → Price
- → Quantity (Initial stock level)

Through the Socket networking interface, on the server side, the coffee.csv file is imported, manipulated, encoded, and sent to the client side of the software system. It is then on the client side which the vending machine will start to take shape, thereby allowing a user to place orders for the coffee products. In addition to providing a graphical user interface for the vending machine, the client side of the software system is responsible to send back information pertaining to each transaction to the client side. These transactions are then manipulated to be stored in a separate .txt file and are also used to update the stock level accordingly. The updated stock is then transmitted to the client side to keep the vending machine up to date.

Server-side Software System

The server-side of the software system is where the entire program will root from. It is the python file in which a connection with the client-side will start and it is also where the coffee.csv file containing data pertaining to products being sold will be imported. It also ensures background operations such as storing each transaction made by a user and received from the client-side into a separate transaction.txt file and updating the stock level to keep the vending machine up to date.

To begin with the program, the following libraries need to be imported:

→ socket
→ csv
→ pickle
→ _thread
→ threading

Each library is essential to the proper running of the program. The socket library is used to ensure the client/server-side software system and act as a point-to-point channel of communication between the client and server. The csv library is needed to be able to read the coffee.csv file. Pickle library is used to encode and decode the data sent to and received from the client-side of the software system. Finally, the _thread and threading libraries are essential for different functions within the program to run concurrently as otherwise, the communication established will terminate once information is sent and received.

After importing the libraries, the coffee.csv file is opened, and the data is converted into a list entitled product_list as shown in the figure below:

```
# Opening the csv file containing all the information needed on the product which the Vending Machine will sell
with open('coffee.csv', 'r+') as csv_file:
    csv_reader = csv.reader(csv_file)   # Using csv module to read the csv file
    product_list = list(csv_reader) # Converting the data inside the csv file into a list
```

Figure 1.

When reading the .csv file and converting its data into a list, the elements formed are in the string format. Hence, since the quantity element would be used for later calculation, the latter was converted into an integer as shown in the following figure:

```
# Converting the quantity of products from the stock from strings to integers for later calculations
for products in product_list:   # Looping through elements of the product list
    x = products[3]     # Extracting the stock quantity of each product (in strings)
    y = int(products[3])    # Converting the quantity of each product into integer
    products[3] = y     # Replacing the string values with the integer values
```

Figure 2.

A thread function is created to ensure that the communication between client and sever remains on-going. It is within this function that the server-side receives the order_list which is a list containing transaction effectuated by the user. This list is then used to save some transaction information into a transaction.txt file and it also used to update the stock level of the products. Once the stock is updated, still within the thread function, the updated stock is sent back to the client-side, ensuring everything remains up to date. This is shown by the following figure:

```
# Function in another thread to be run concurrently with main function
def threaded(conn):
    while True:     # Creating an infinite loop
        order_details = pickle.loads(conn.recv(1024))   # Receiving the order list as transaction from client side

        for product in product_list:    # Looping through elements of the product list
            for details in order_details:   # Looping through elements of the order list
                if details[0]==product[0]:  # Matching element from the product list to the order list using their
                    # product ID
                    # Creating and opening up a transaction .txt file to store each transaction
                    with open('transaction.txt', 'a') as transaction_file:
                        transaction_file.writelines(f'Item purchased: {details[1]}\nQty purchased: {details[3]}\n\n')
                    # Updating the stock of the products by decreasing the quantity available by the quantity already
                    # ordered
                    stock = product[3] - details[3]
                    product[3] = stock # Replacing the quantity available in the product list with the updated stock
                    # level quantity
        # Encoding the updated stock using pickle for better data transmission
        updated_stock = pickle.dumps(product_list)
        # Sending back an updated version of the stock to the client
        conn.send(updated_stock)
```

Figure 3.

The main function is the one which creates the connection by setting up a host, a port, the address family, and the TCP protocol. This function also permits the server to listen to different client connections and can put some in waiting as well. The original product_list is sent to the server in this function and the thread function is called to be run concurrently. This is by the following figure:

```python
# Main function of the server which establishes connection with client
def main():
    host = 'localhost'  # Declaring a host
    port = 12345    # Declaring a port on which to operate
    # Create a server socket
    # AF.INET refers to the address family IPV4
    # SOCK_STREAM means connection oriented TCP protocol
    server_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Binding server socket to the host and port
    server_client.bind((host, port))
    print("socket bound to port", port)

    # put the socket into listening mode
    server_client.listen(5)
    print("socket is listening")

    # a forever loop until client wants to exit
    while True:
        # establish connection with client
        conn, addr = server_client.accept()

        print('Connected to :', addr[0], ':', addr[1])

        # Encoding product list using pickle for better data transmission
        data = pickle.dumps(product_list)
        conn.send(data) # Sending the product list to the client

        # Start a new thread and return its identifier
        start_new_thread(threaded, (conn,))


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

Figure 4.

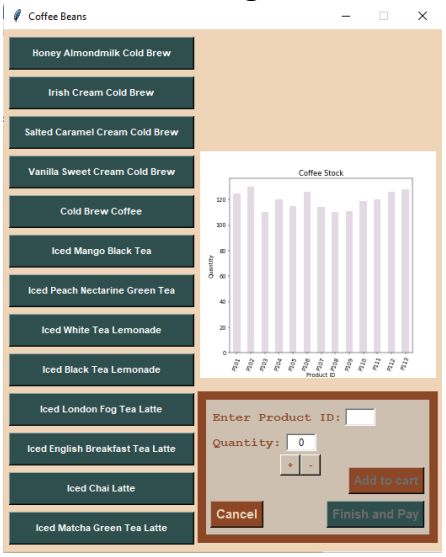
Client-side Software System

The client-side of the software system is where the majority of the code lies. This is where the GUI is implemented as well as Matplotlib. As with the server.py file, before beginning to code, certain libraries need to be imported which will allow the implementation of the Tkinter and Matplotlib modules. Hence, in this client.py file, the following libraries and modules were imported:

→ socket
→ tkinter
→ messagebox from tkinter
→ DataFrame from pandas
→ Matplotlib.pyplot
→ FigureCanvasTkAgg from matplotlib.backends.backend_tkagg

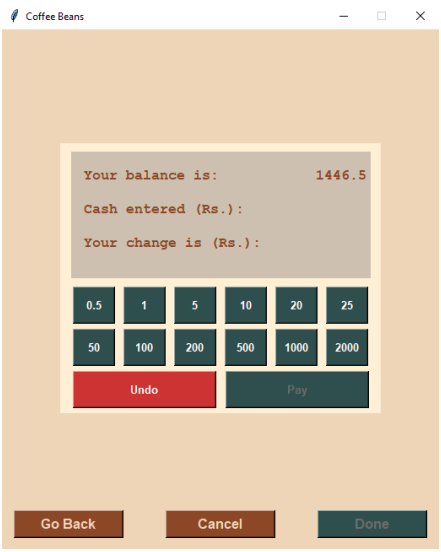
Socket is used to create to point-to-point connection with the server-side of the software system. Tkinter was used to create the graphical user interface of the vending machine while messagebox was used to display a warming message if a user tried to order more of a product than the stock available. DataFrame was used alongside Matplotlib.pyplot and FigureCanvasTkAgg to create a bar chart providing visual presentation of the stock available for the different products as well as embedding said bar chart into the tkinter GUI.

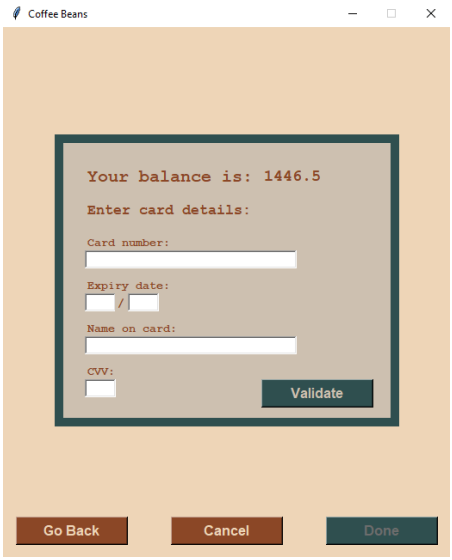
When launched, the vending machine boots up with a welcoming message and prompts the user to the Menu page whereby he/she can start ordering. The following figures showcase how the vending machine works:

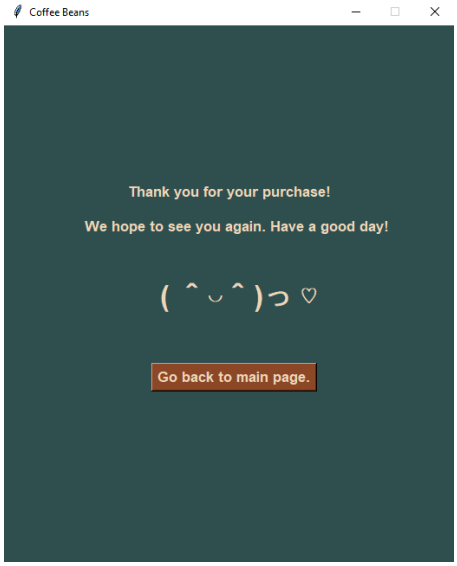


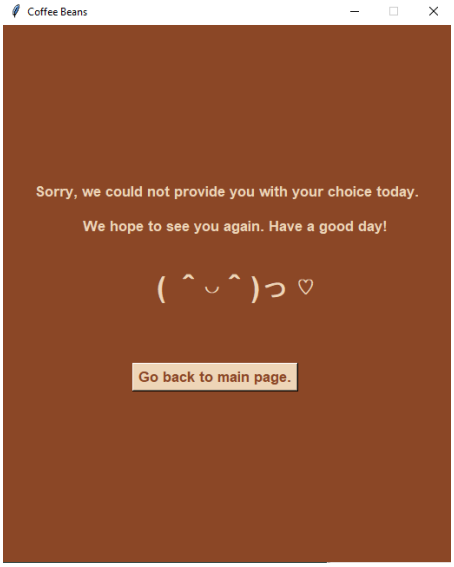

Coffee Beans
========== Your Receipt ==========
Description Qty Cost Price
Cold Brew Coffee 2 100 200.0
Iced English Breakfast Tea Latte 3 175.5 526.5
Iced Matcha Green Tea Latte 3 160 480.0
Iced Chai Latte 2 120 240.0
Total 1446.5
Go Back
Cancel
Payment Method

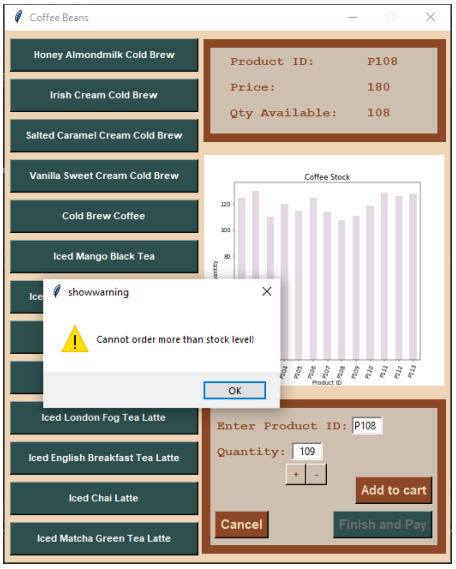

Coffee Beans
(≈ง ᵒ̌)
Choose your payment method:
Cash
Card
Go Back
Cancel

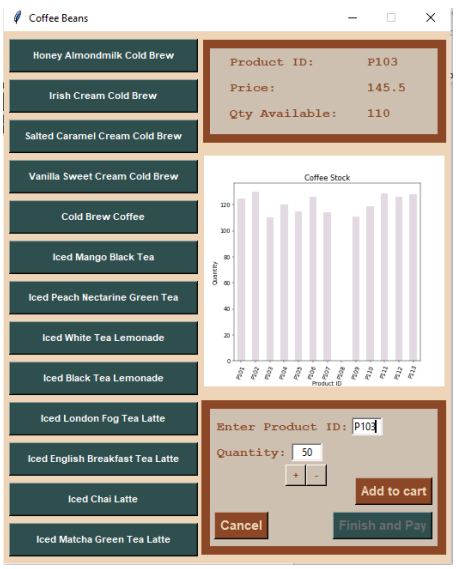

Coffee Beans
Your balance is: 1446.5
Cash entered (Rs.):
Your change is (Rs.):
0.5 1 5 10 20 25
50 100 200 500 1000 2000
Undo
Pay
Go Back
Cancel
Done

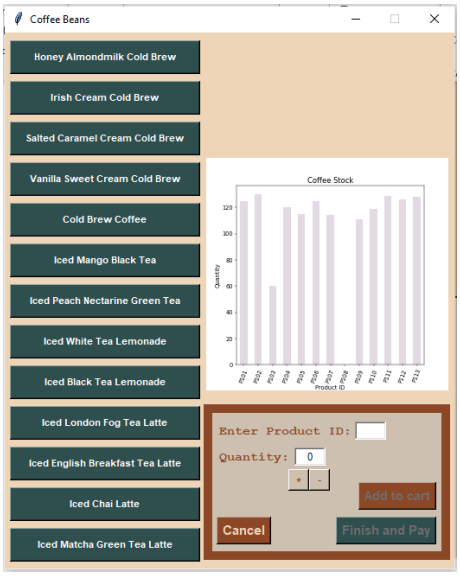

Coffee Beans
Your balance is: 1446.5
Enter card details:
Card number:
Expiry date:
/
Name on card:
CVV:
Validate
Go Back
Cancel
Done


Coffee Beans
Thank you for your purchase!
We hope to see you again. Have a good day!
(^ ᵕ ^)つ ♡
Go back to main page.


Coffee Beans
Sorry, we could not provide you with your choice today.
We hope to see you again. Have a good day!
(^ ᵕ ^)つ ♡
Go back to main page.


Coffee Beans
Honey Almondmilk Cold Brew
Irish Cream Cold Brew
Salted Caramel Cream Cold Brew
Vanilla Sweet Cream Cold Brew
Cold Brew Coffee
Iced Mango Black Tea
Iced London Fog Tea Latte
Iced English Breakfast Tea Latte
Iced Chai Latte
Iced Matcha Green Tea Latte
Product ID: P108
Price: 180
Qty Available: 108
Coffee Stock
showwarning
Cannot order more than stock level!
OK
Enter Product ID: P108
Quantity: 109
Add to cart
Cancel
Finish and Pay

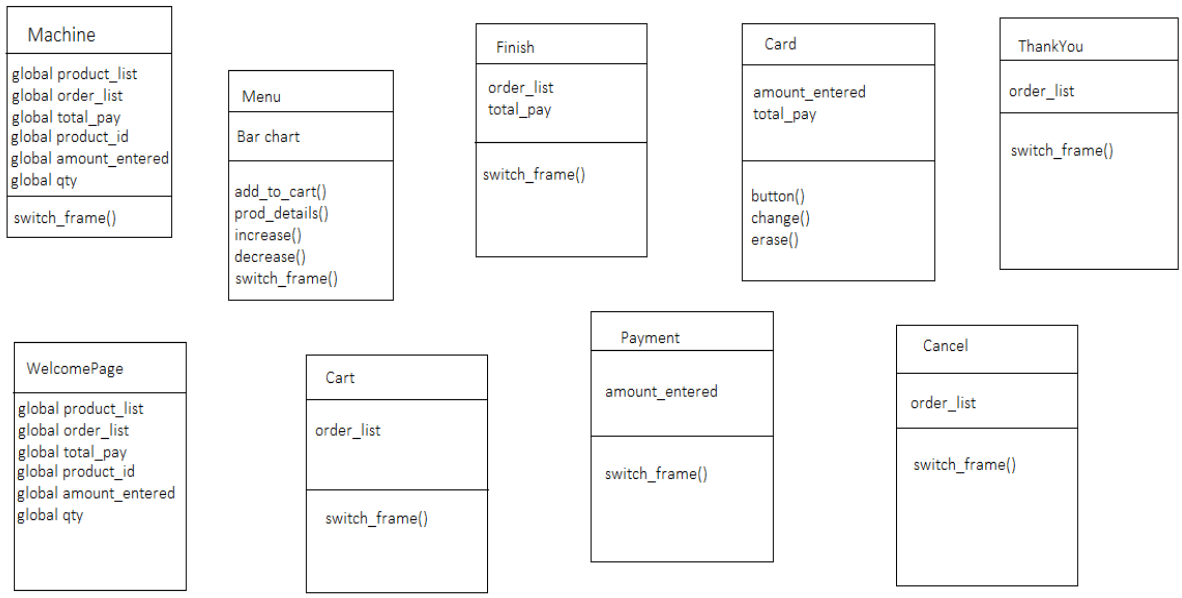

Coffee Beans
Honey Almondmilk Cold Brew
Irish Cream Cold Brew
Salted Caramel Cream Cold Brew
Vanilla Sweet Cream Cold Brew
Cold Brew Coffee
Iced Mango Black Tea
Iced Peach Nectarine Green Tea
Iced White Tea Lemonade
Iced Black Tea Lemonade
Iced London Fog Tea Latte
Iced English Breakfast Tea Latte
Iced Chai Latte
Iced Matcha Green Tea Latte
Product ID: P103
Price: 145.5
Qty Available: 110
Coffee Stock
Enter Product ID: P103
Quantity: 50
Add to cart
Cancel
Finish and Pay


Coffee Beans
Honey Almondmilk Cold Brew
Irish Cream Cold Brew
Salted Caramel Cream Cold Brew
Vanilla Sweet Cream Cold Brew
Cold Brew Coffee
Iced Mango Black Tea
Iced Peach Nectarine Green Tea
Iced White Tea Lemonade
Iced Black Tea Lemonade
Iced London Fog Tea Latte
Iced English Breakfast Tea Latte
Iced Chai Latte
Iced Matcha Green Tea Latte
Coffee Stock
Enter Product ID:
Quantity: 0
Add to cart
Cancel
Finish and Pay

When the user tries to order more than the stock available, a warning message pops up and the add to cart button as well as finish and pay are disabled. After completing a transaction, the bar chart updates with the new stock available, and the user can see this through the visual presentation Matplotlib offers. After each order, the user can proceed, cancel, or go back to order more products. The Finish and Pay button prompts to a receipt whereby the user sees all their orders listed down have a choice to go back or move on to payment methods. If the user decides to cancel, they are prompted to an apology message and then back to the welcome page. Once they finalize the transaction, they are prompted to a thank you message and back to the welcome page.

UML Diagram

Below is the UML diagram for the whole vending machine:



Improvement and Conclusion

There are several ways in which this vending machine program could be improved. The interface could be rendered more user friendly by:

→ Allowing user to edit quantity ordered of each order one by one.
→ Option for potential refunds.
→ The card payment could have an actual card verification instance.

Working on this coursework was very interesting and at times challenging. Certain limitations were encountered but they were rapidly resolved through some personal extensive research. The biggest limitation encountered was while running the client/server-side software system. Multithreading had to be implemented to ensure that the programs function would run concurrently, and the communication would not terminate. However, at the end of the day, the projects wraps up well and offers a visually appealing and intuitive coffee vending machine.