Coursework 2: Smart Vending Machine for Digital Goods

CST1510 Programming for Data Communication and Networks

Credentials:

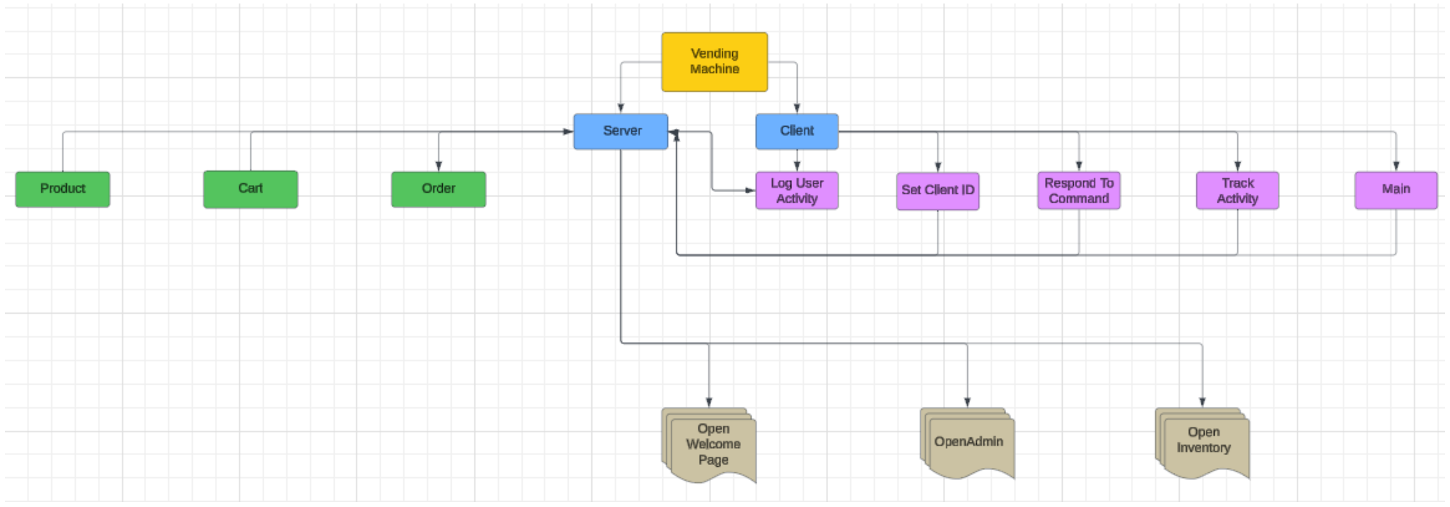
Haydn Campbell

M01001693

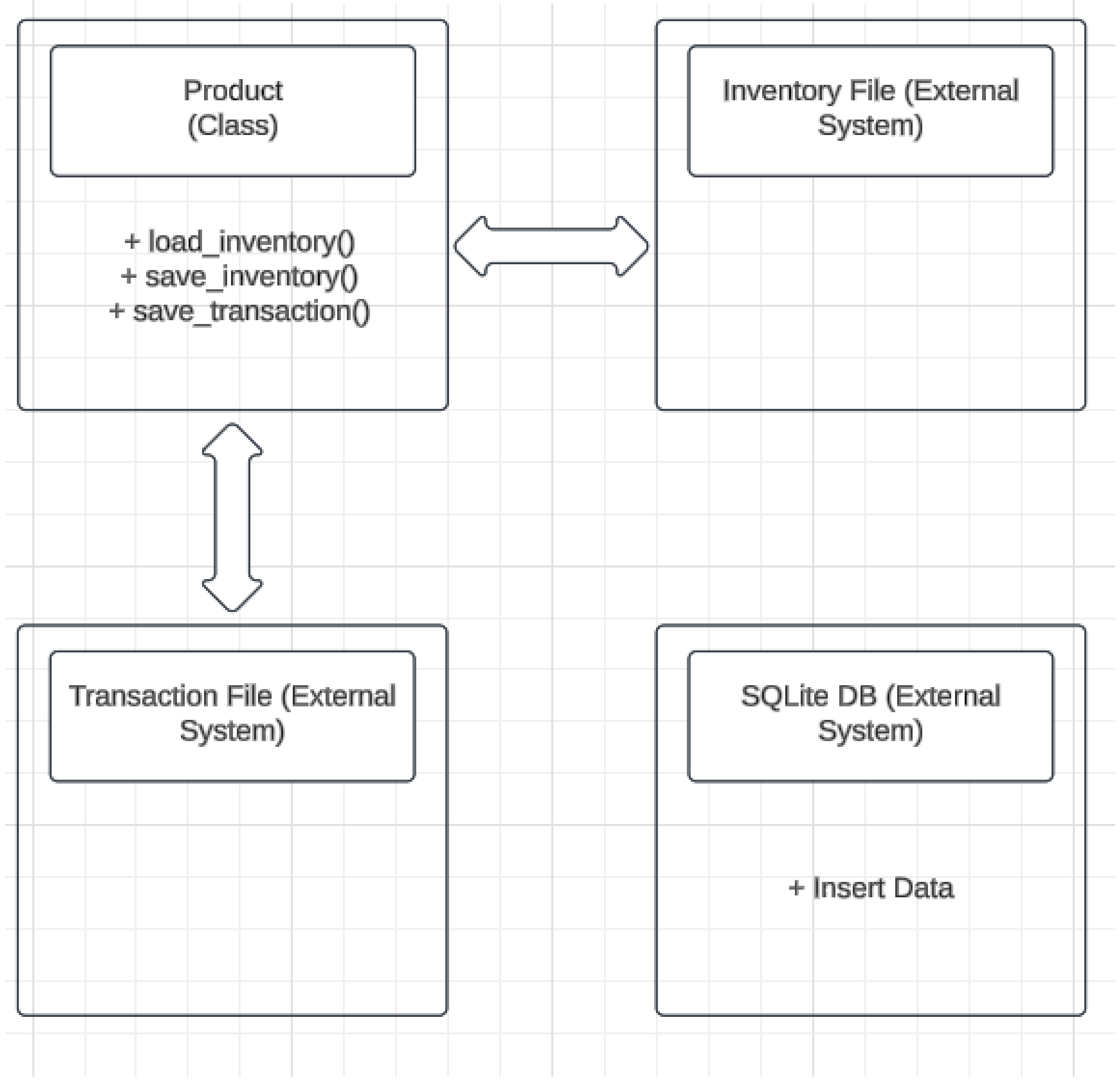
Contents Page:

UML Diagrams

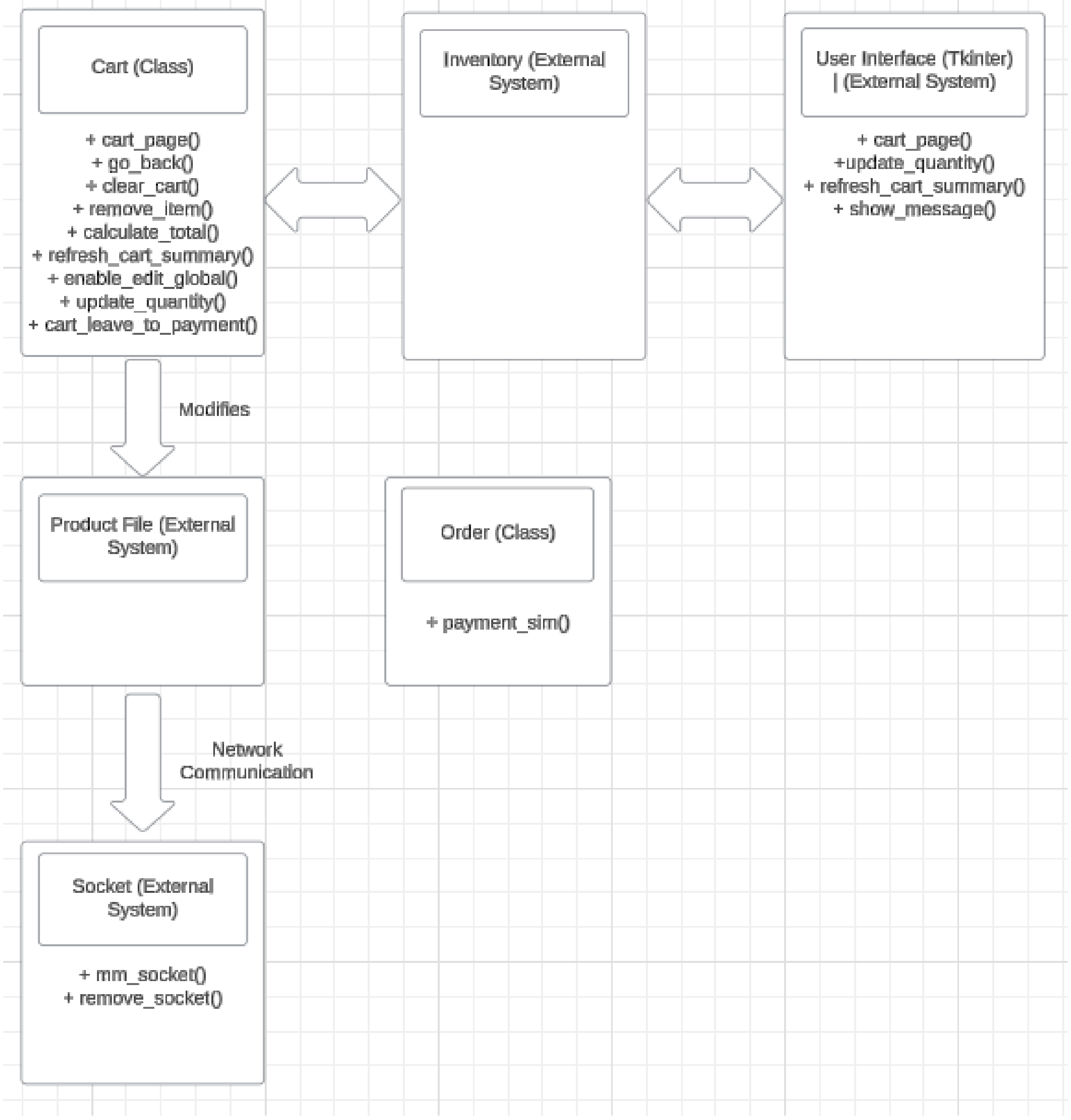
Vending Machine UML Diagram (Of Classes and Functions)



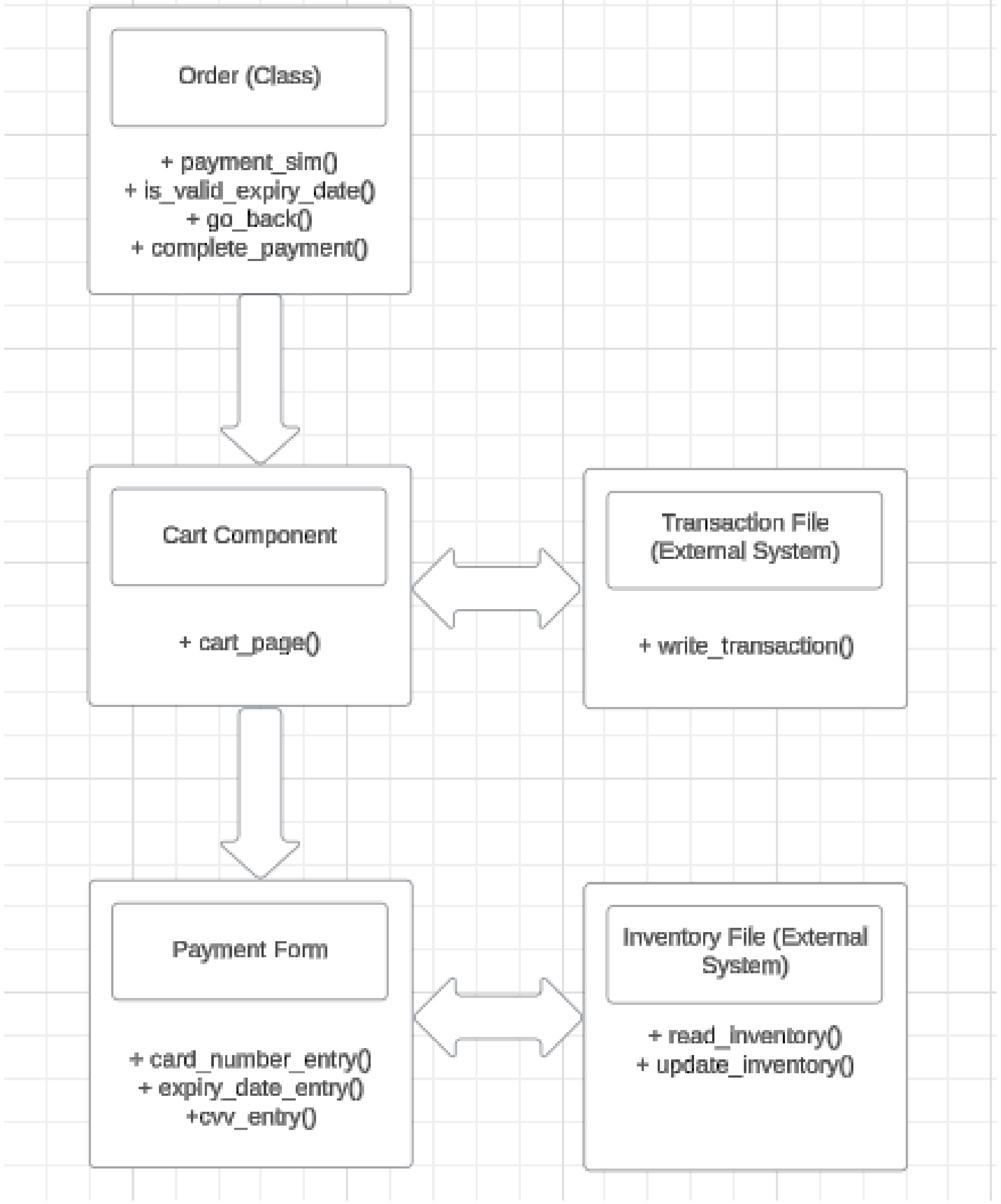
Product Class UML Diagram


Product
(Class)
+ load_inventory()
+ save_inventory()
+ save_transaction()
Inventory File (External System)
Transaction File (External System)
SQLite DB (External System)
+ Insert Data

Cart Class Component UML Diagram



Order Class Component UML Diagram



Implementation Plan

At the beginning of the project, I decided to focus primarily on enhancing user interactivity with the main program, ensuring that as much of the application as possible was user oriented. This decision required significant changes to the code, particularly when implementing object-oriented programming (OOP), client-server architecture, and the Tkinter user interface. The OOP implementation posed challenges in grouping related content together and ensuring that instances were properly called when needed within specific classes. One notable difficulty arose with the Cart class, where I aimed to allow customers to modify the quantity of items in their cart before reaching the final checkout stage. This involved managing multiple calls to the inventory and fresh_inventory text files to perform necessary comparisons, ensuring there were no logical errors and preventing customers' actions from inadvertently altering the database.

During the transition to server-client code, the connection setup didn’t go as smoothly as anticipated, and some logical errors remain in the code. Currently, the system is intended to track user and client page access during their interaction with the vending machine program. However, combining Tkinter, SQLite, and server-client programming has introduced challenges. For example, an issue in the code is that `sqlite.connection` does not allow the use of the `sendall` function to transmit data back to the client code page. This means that, although the client and server are connected, commands and responses cannot be exchanged between them due to limitations in function support across the implementation areas.

The Tkinter section of the project was invaluable for customizing the system and providing a unique experience for users. The graphical interface simplified navigation, making it as intuitive as the previous text-based system. The system as a whole is structured around managing the `inventory.txt` file, which is responsible for importing, appending, updating, and removing data whenever relevant commands are executed or when the code is run. Additionally, the `transaction.txt` file is used to record completed transactions, saving essential details such as the card number, client ID, CVV, card expiry date, transaction ID, card type, cart items, quantities, and the total cost. Together, these two text files form the foundational structure of the virtual vending machine system.

Running Instructions

1. Download the following files from the CST1510Project.zip file: `MAINSERVER.py`, `MAINCLIENT.py`, `inventory.txt`, `fresh_inventory.txt`, `transaction.txt`, and the Images folder. Note that the `vending_machine.db` file will be created upon compilation to reduce wait time on the first run.
2. Once the files are downloaded, open the `MAINSERVER.py` file in the command prompt (CMD). Then, open the `MAINCLIENT.py` file in a separate CMD tab.
3. When both are running, the Tkinter page will appear. To view the full page, click the expand tab in the top right corner.
4. To access the inventory and add items to your cart, click the "View Inventory" button (green). When the page loads, scroll down and click on the 'Fishing Virtual Manual'.
5. Add any other product to your cart by clicking its square in the grid.
6. After adding both products to the cart, click the cart button in the top right corner. When the cart page loads, click the "Remove" button next to the item you want to remove.
7. Next, click the "Edit Quantity" button next to the 'Fishing Virtual Manual' item in the cart. A separate box will appear below the 'Fishing Virtual Manual' item.
8. In this box, delete the value '1' and enter '2', then click "Save".
9. After pressing "Save," a pop-up will appear confirming that the quantity has been changed. Click "OK" on the pop-up and then click the "Complete Order" button.
10. Once you click "Complete Order," you will be redirected to the payment completion page. Enter your payment details. There are multiple checks in place to ensure that the information entered is correct. (You can test different entries to check for validation.)
11. After entering the correct details, click the "Complete Order" button and click "OK" on the completed order pop-up.
12. Now, click "Admin" (blue) and then click "Admin Login."
13. On the login page, enter the username "admin" and the password "password." After typing these into their respective boxes, click "Login."
14. Once logged in, you should see "Fishing Virtual Manual" displayed with the "Refill Stock" button to the right of the page.
15. Click "Refill Stock" and then click the "Logout" button.
16. After logging out you will be returned to the home page, click "View Inventory," and scroll to the bottom of the page.
17. Hover over the 'Fishing Virtual Manual' item to see that its stock has been refilled to 2.

Reflection

Developing this project was a challenging yet rewarding experience that significantly expanded my understanding of Python programming. The inventory system was the backbone, teaching me the importance of managing data dynamically through file operations and ensuring consistency across the system. Working with `inventory.txt` and `transaction.txt` files, I learned to handle real-time updates and avoid logical errors, especially when syncing with the cart and order processes.

Refactoring the code into an object-oriented structure was difficult but improved organization. Managing interactions between classes like `Cart` and `Order` was particularly tricky when ensuring smooth updates to item quantities or validating inputs during the checkout process. The client-server model added another layer of difficulty, especially when socket communication clashed with SQLite connections. Debugging issues like the incompatibility of `sqlite.connection` with `sendall` taught me about system integration and error handling.

Designing the GUI with Tkinter was rewarding, as it transformed the system into a user-friendly platform. Features like editing cart quantities dynamically and processing payments demonstrated the importance of clear workflows and responsive design.

If I had more time, I'd focus on advanced error handling, improving client-server interactions, enhancing the interface with modern frameworks and making sure that commands from the server received responses from the client regardless of its calling position within the code. Overall, this project was an excellent opportunity to apply theory to a real-world scenario and expand my Python programming knowledge.