CPEN 333

Section 101

AMAZOOM Final Project

Project Group 44:

Gurman Toor 77651546

Gurmukh Hare 62400726

Akash Randhawa 88346861

Table of Contents

Executive Summary……3

Design Justifications……4

Communications Protocol……4

UML Diagrams……5

Use Case Diagram……5

Sequence Diagram……8

Class Diagram……9

Object Interaction Diagram……9

Communication Diagram……10

Executive Summary

In the year of 2020 while many around the globe were stuck in their homes due to the pandemic, “over 2 billion people purchased goods or services online” [statista.com]. It is evident that ecommerce platforms and online retail websites are growing at rapid rates as more consumers look to use the internet to make purchasing products easier from the comfort of their homes. We believe that we have developed a solution that maximizes the use of innovative technology and automation to optimize the experience for customers in a way that no other ecommerce platform on the market does. This is where Amazoom-44, an ecommerce platform developed by Project44 Inc, comes into the picture.

Amazoom-44 utilizes advanced robots in the warehouse that possess object recognition, automated navigation, and collision avoidance capabilities. The software system consists of a remote webserver which accepts orders from multiple clients through sockets setup using TCP. The client interface allows the consumers to view products and their cart, add/remove items in their cart, checkout their order/view the price, and exit the store after placing an order. The data exchanged between the client and server is in JSON format, which is serialized/deserialized as needed on each end. As multiple clients interface with the webserver, the catalogue and stock of our products is updated and the changes can be observed on each client. In addition to the client, there is an admin panel that allows the admin of the warehouse to restock inventory as needed, query the status of orders, view inventory, and view alerts. Both the admin panel and client interfaces are programmed using C# and run on separate CLI terminal windows.

When orders are received at the webserver, they are sent to our backend for processing. Orders are first validated to ensure that they can be fulfilled. We then run our robots on separate threads to retrieve items from the warehouse inventory and fulfill the order. Mutual exclusion is used for our warehouse so that only one robot can access a grid in our warehouse to retrieve an item. This avoids collisions amongst robots as they collect their assigned items. These robots are dynamically assigned as required, depending on the number of items in the order. Once orders are fulfilled, the server is responsible for sending our delivery trucks as needed. If multiple orders are received within a specified timespan, they will be allocated to the same delivery truck if the weight capacity is not exceeded. Otherwise, a new truck is assigned. As items in our inventory run low, restocking trucks replenish our stock and alerts are provided to the admin.

We strongly believe that Amazoom-44 is the most optimal solution since the system boils down to a very simple design that still satisfies all the requirements. Because the system’s tech stack is very consistent front to back (C#), adding new features, debugging the system, and scaling the system is straightforward. The system accommodates for various sizes of warehouses and assigns resources (robots) as required. The system is designed for simplicity and efficiency, so Project44 Inc believes that this is exactly the solution that your company is looking for.

Design Justifications

Robot:

- Robots have advanced pathfinding methods already implemented.
- Collision avoidance is therefore only implemented when occupying a spot in the warehouse, not for moving around.

Product Database:

- Used a Json file for product “database” to simplify the serialization of data between different processes as well as keeping our implementation simple and effective
- Used two separate json files, inventory, and catalogue, because the products seen by the warehouse and the products seen by the clients have different values associated with them.
- There are two classes, product, and item. Product is used on the front end to show the clients what is in stock. Item is used in the backend, so the warehouse knows exactly how many of each product are in the warehouse and distinguish each one.

Trucks:

- The Delivery truck will wait for a specified time-period before leaving with completed orders even if it is not full. This is done so that if there are no new orders being placed for a long time by clients, the truck does not sit at the warehouse and instead delivers the orders to clients maximizing usage of trucks.
- When the admin discontinues a product, all items of it needs to leave the warehouse. This is done in a separate delivery truck that does not mix in regular deliveries, so space is freed up sooner.

Stocking:

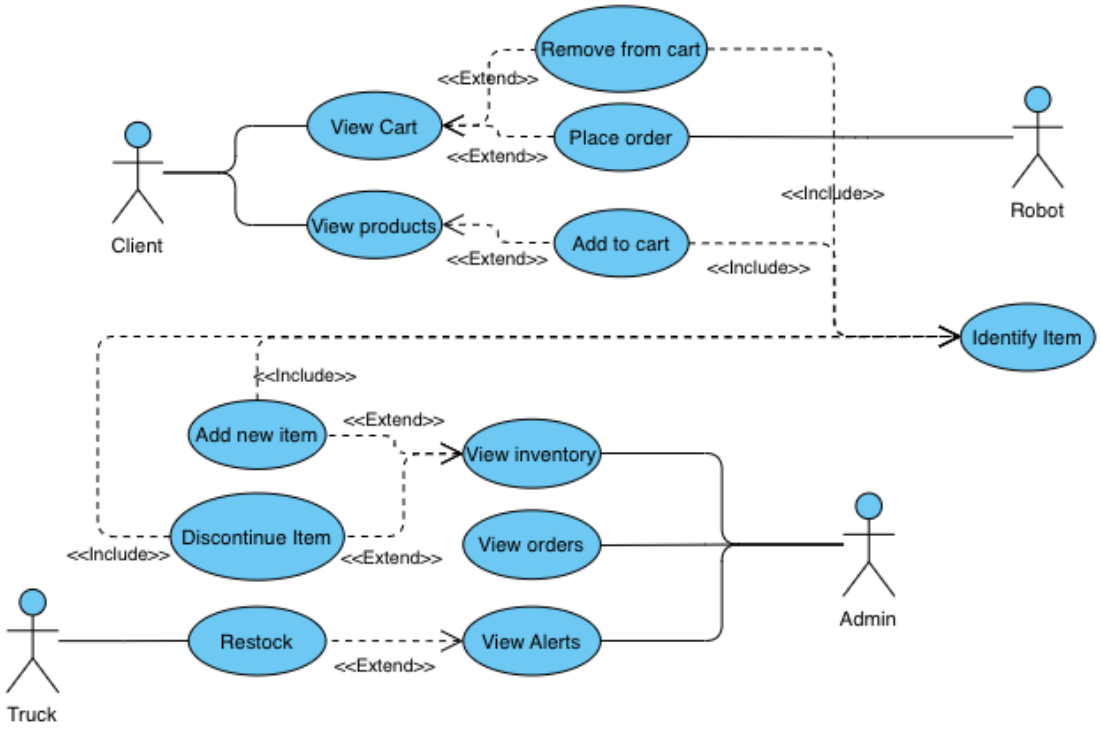
- When an item is out of stock, it will alert the admin which can then replenish stock. Upon replenishment, not only is the out-of-stock item restocked but every item below max capacity. This is done to keep the warehouse full and efficient.

Communication Protocol:

The Client-Server communication is backed by the TCP protocol. The server is able to accept multiple client socket connections. Once a connection has been accepted, the client can send a command. Once the command message is decoded from a byte stream to a string, the server then processes the request. The server is able to send data back to the client through the open socket. The commands available are for receiving JSON inventory data from the server, decrementing stock in the inventory, incrementing stock in the inventory and sending an order to be fulfilled and delivered.

UML

USE CASE Diagram and Scenario



CLIENT:

View Cart: (shows cart and cart menu option)

1. Establish a connection with Server
2. Read all products in database from server
3. Display only the products in user's cart
4. If user wants to remove an item
   Extend Use case: remove item from cart
5. If user wants to checkout
   Extend use case: place order

Remove Item: (removes an item from users' cart)

1. If cart is already empty <<Scenario 1>>
2. Include *Identify Item*
3. Remove item from the user's cart
4. Update the catalogue to cancel the reserve on the product
5. Redisplay the cart

Scenario 1: Prompt user that cart is empty. Redisplay all products.

Place Order: (sends order to server)

1. Establish a connection with Server
2. Read all products in database from server
3. Display the total price of the cart using server data
4. If user decides to confirm the order, send the order to remote server
5. Clear cart and redisplay the store

View Products: (shows all available products and menu options)

1. Establish a connection with Server
2. Read all products in database from server
3. Display all products in the catalogue
4. If user wants to add an item to their cart:
   Extend use case: add item to cart
5. Reconnect and display products with updated inventory

Add Item: (adds an item to users' cart)

1. If cart is already full<<Scenario 1>>
2. Include *Identify Item*
3. Add item to the user's cart
4. Update the catalogue to reserve the product added
5. Redisplay the cart

Scenario 1: Prompt user that cart is full. Ask them to checkout.

ADMIN:

View Inventory: (view warehouse product inventory and menu options)

1. Get all products in the inventory
2. Display all the products in the inventory
3. If user wants to add a new item
   Extend: Add new Item
4. If user wants to discontinue an item
   Extend: Discontinue item

Add New Item: (adds new item to warehouse)

1. Include *Identify Item*
2. Add product to warehouse catalogue
3. Redisplay Inventory

Discontinue Item: (removes item from warehouse for good)

1. Include *Identify Item*
2. Remove item from warehouse catalogue
3. Add discontinued status to product
4. Redisplay Inventory

View Orders: (view all placed and in process orders)

1. Get order log from warehouse
2. Display all orders and their status code

View Alerts: (view out of stock alerts)

1. Get all products in the inventory
2. Check if products have gotten an out-of-stock alert
3. Display all out-of-stock products and status message
4. If user wants to replenish stock
   Extend: Restock
5. Clear alerts

Restock: (restock all products below max capacity)

1. Read product catalogue and check what is below max capacity
2. Send trucks to bring new inventory to the warehouse
3. If Trucks fail prompt user that they will be resent
4. Trucks restock warehouse

BOTH:

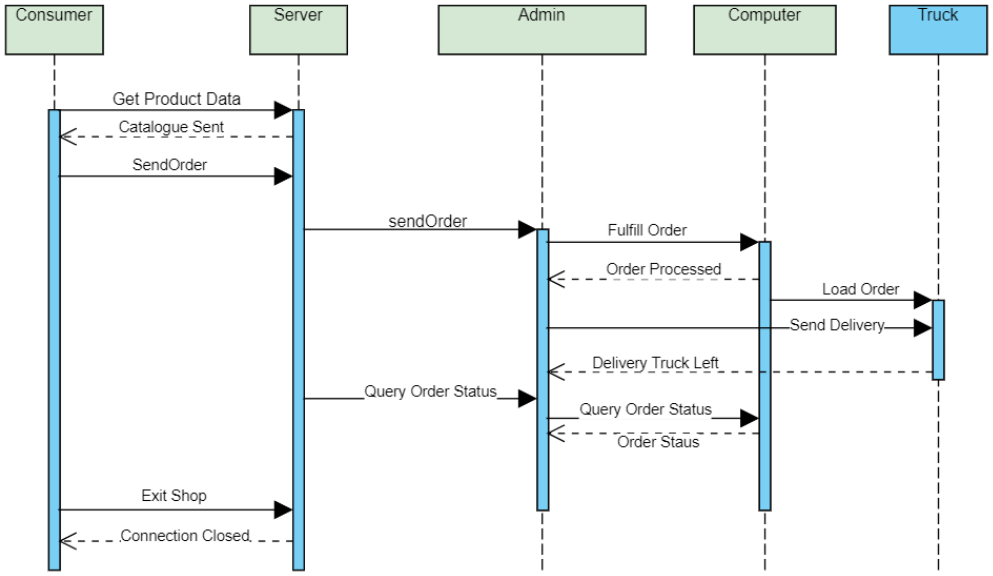
Identify Item: (find the item the user is wishing to locate)

1. Prompt user to input product identifier
2. If user enters invalid input <<Scenario 1>>
3. Return the product item that is being identified

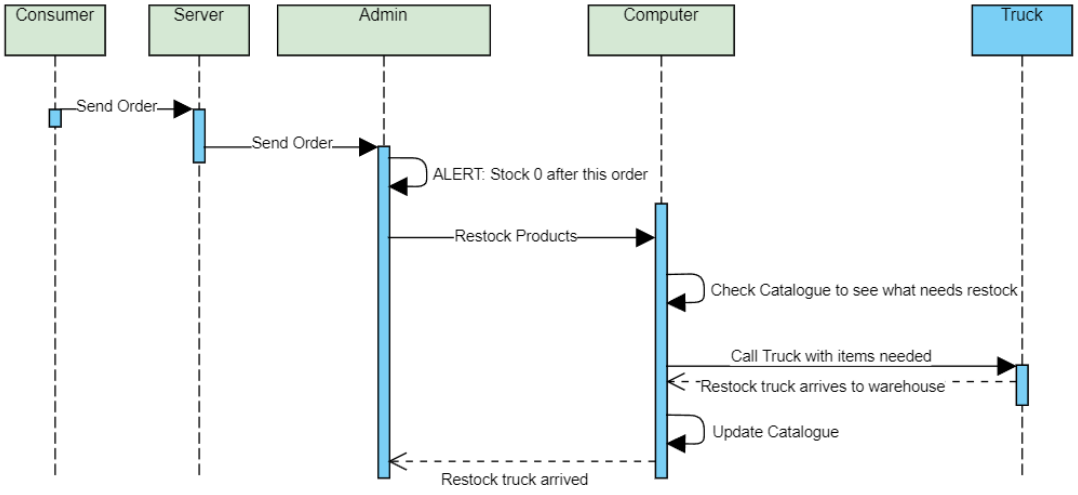
Scenario 1: Re-prompt user to enter a valid input until they do so.

Sequence Diagrams

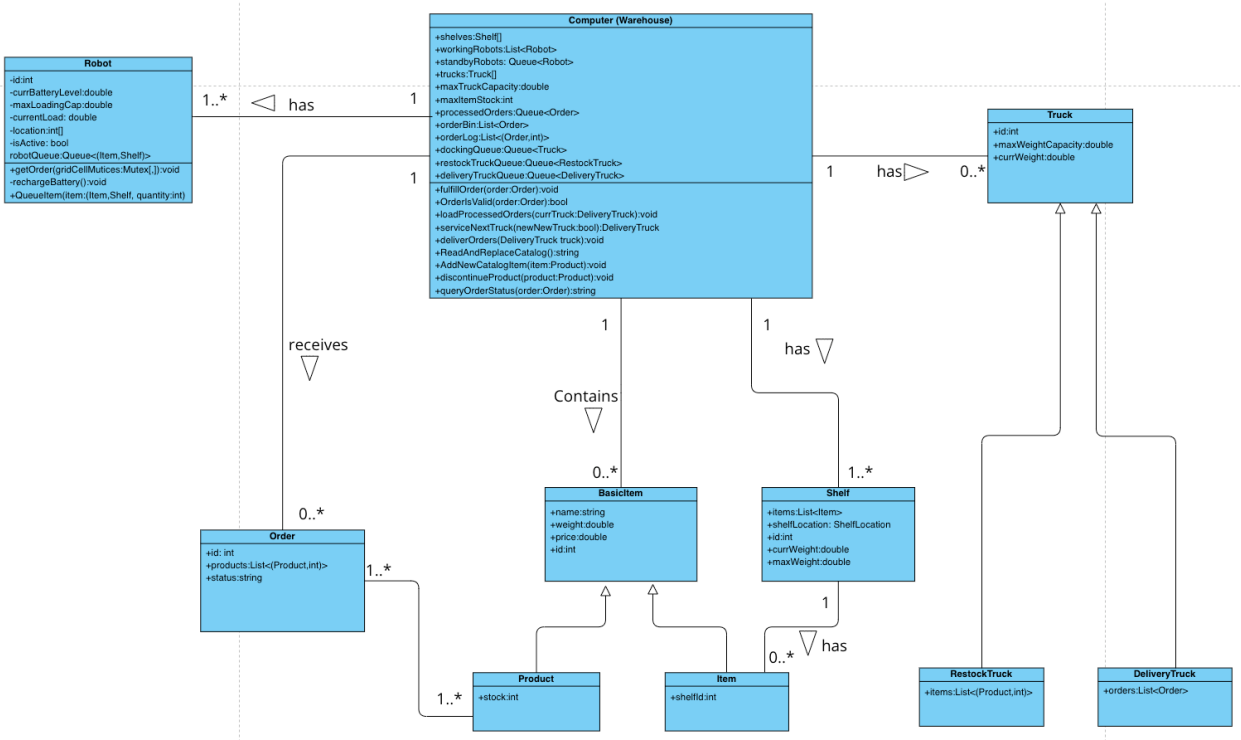
Client Ordering an item:



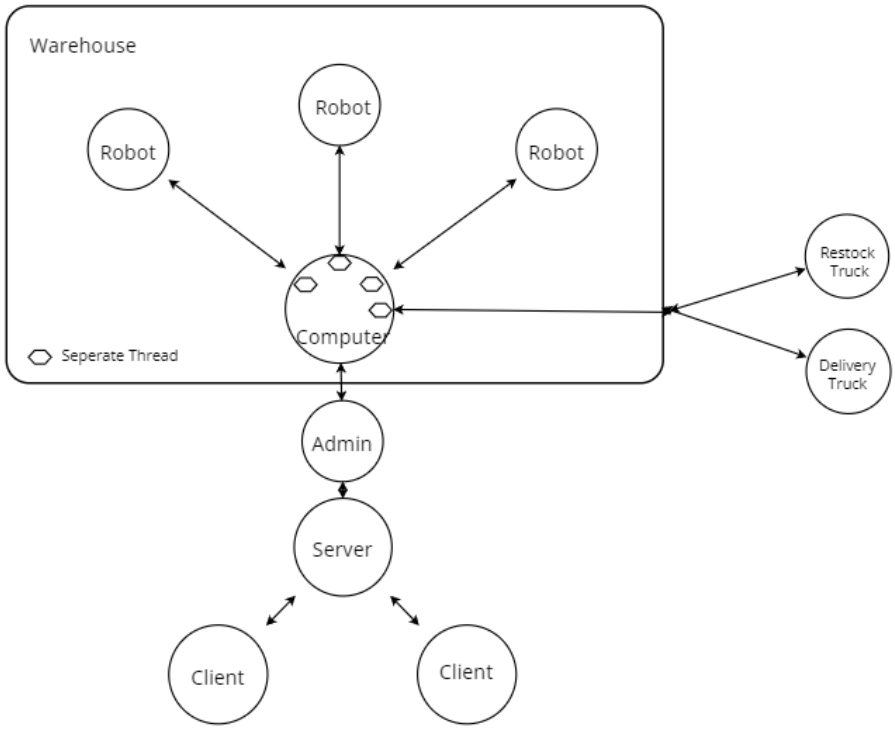
Client ordering the last stock to initiate restock:



Class Diagram



Object Interaction Diagram



Communication Diagram