Amazoom

•••

By: Gurmukh Hare, Akash Randhawa, Gurman Toor

Agenda

1. Design Overview
2. Live Demo
3. Testing
4. Questions
5. Appendix

Design Overview

Client Side:

- Client interaction through CLI windows
- Communicate with server through TCP sockets
- Multiple clients allowed to run simultaneously
- Clients exchanges JSON data with server

Server Side:

- Robots run on separate threads for order item retrieval
- Mutual exclusion for grid locations in warehouse to avoid robot collisions and for reading and writing to inventory
- Delivery trucks sent out on a timed interval if no new orders received from clients or when full

Live Demo

Testing

- Separate testing class to break down our workflows and validate functionality
- Tested workflows include:
  1. TestAddCatalogProducts()
  2. TestRestockInventory()
  3. TestOrderValidation()
  4. TestFulFillOrder()
- Manual testing in addition to automated tests also completed

Questions?

Thank you!

●●●

APPENDIX

Design Justifications & Assumptions

Design Justifications and Assumptions:

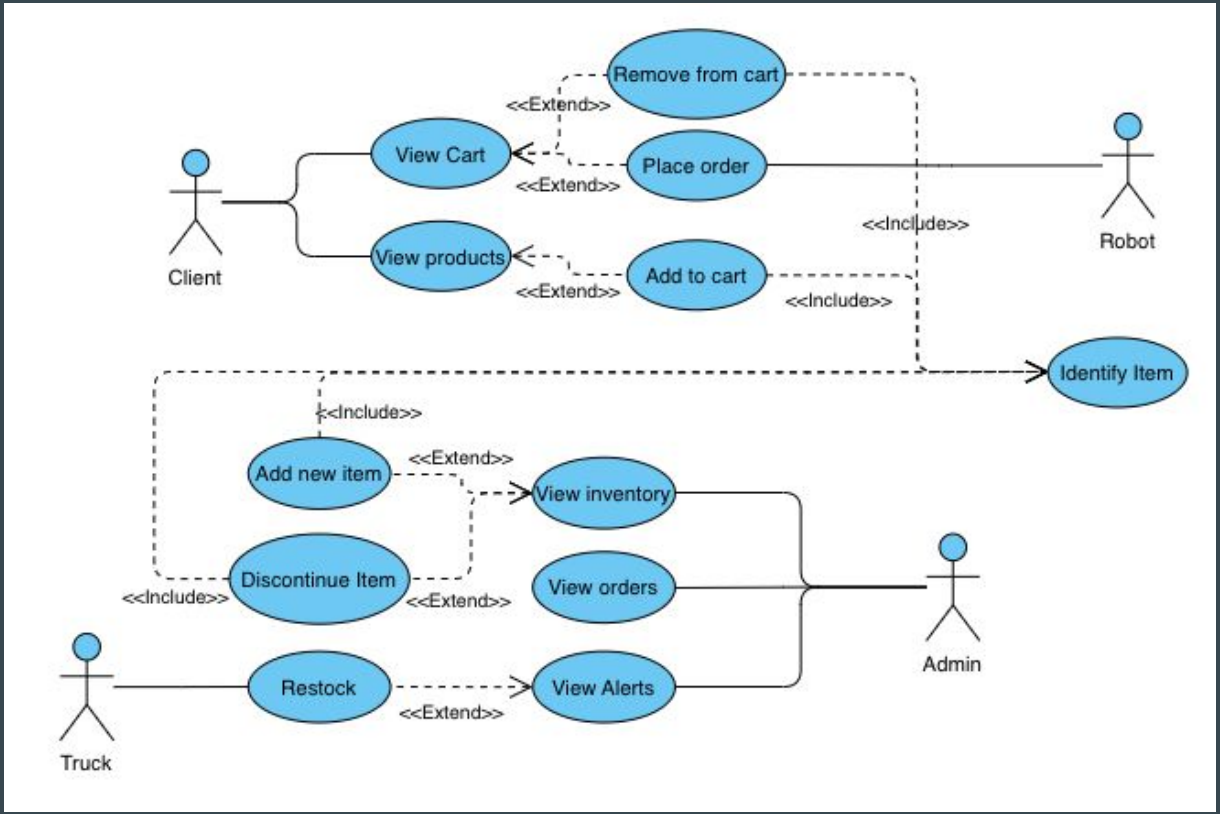
Robot:
- Robots have advanced pathfinding methods already implemented.
- Collision avoidance is therefore only implemented when occupying a spot in the warehouse, not for moving around.

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.

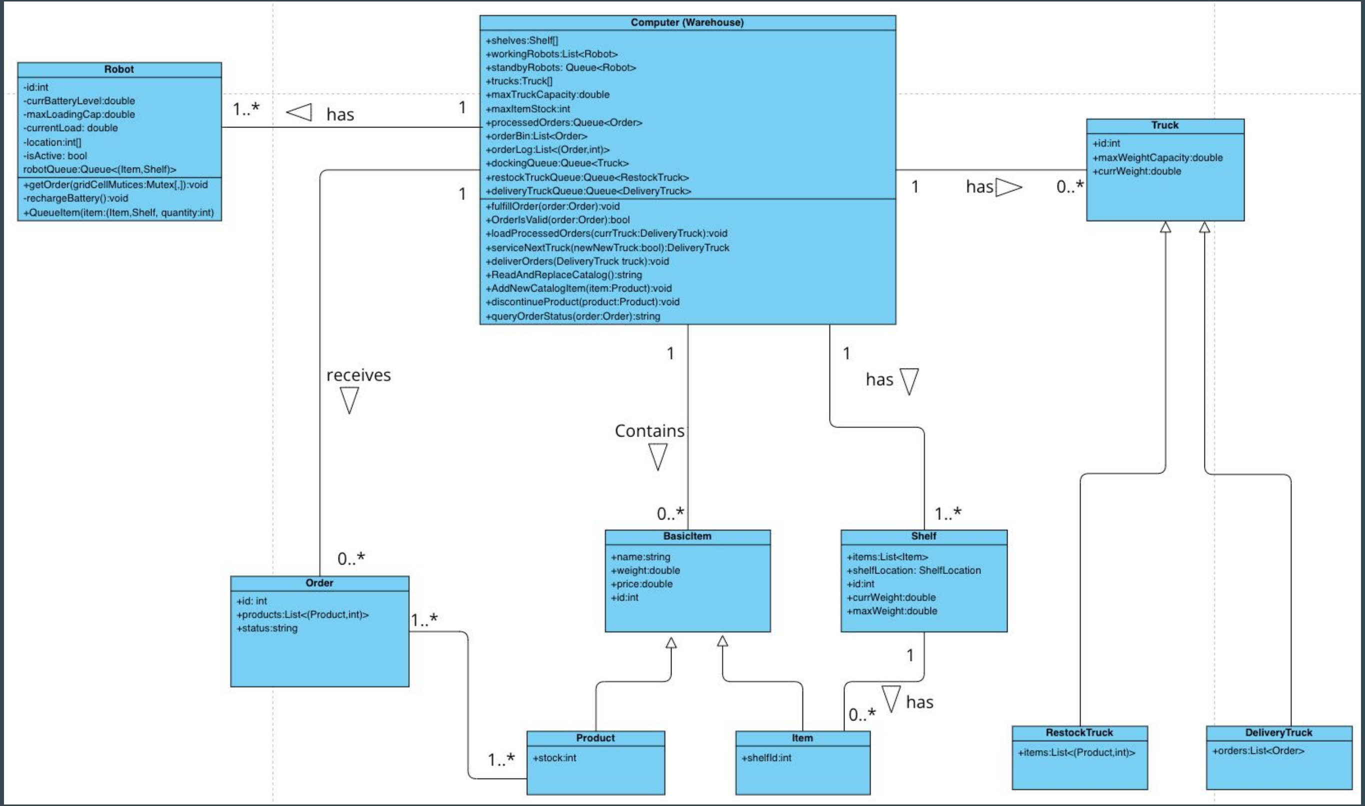
Trucks:
- The Delivery truck will wait for a specified time-period before leaving with completed orders even if it is not full. This is done so that if there are no new orders being placed for a long time by clients, the truck does not sit at the warehouse and instead delivers the orders to clients maximizing 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.

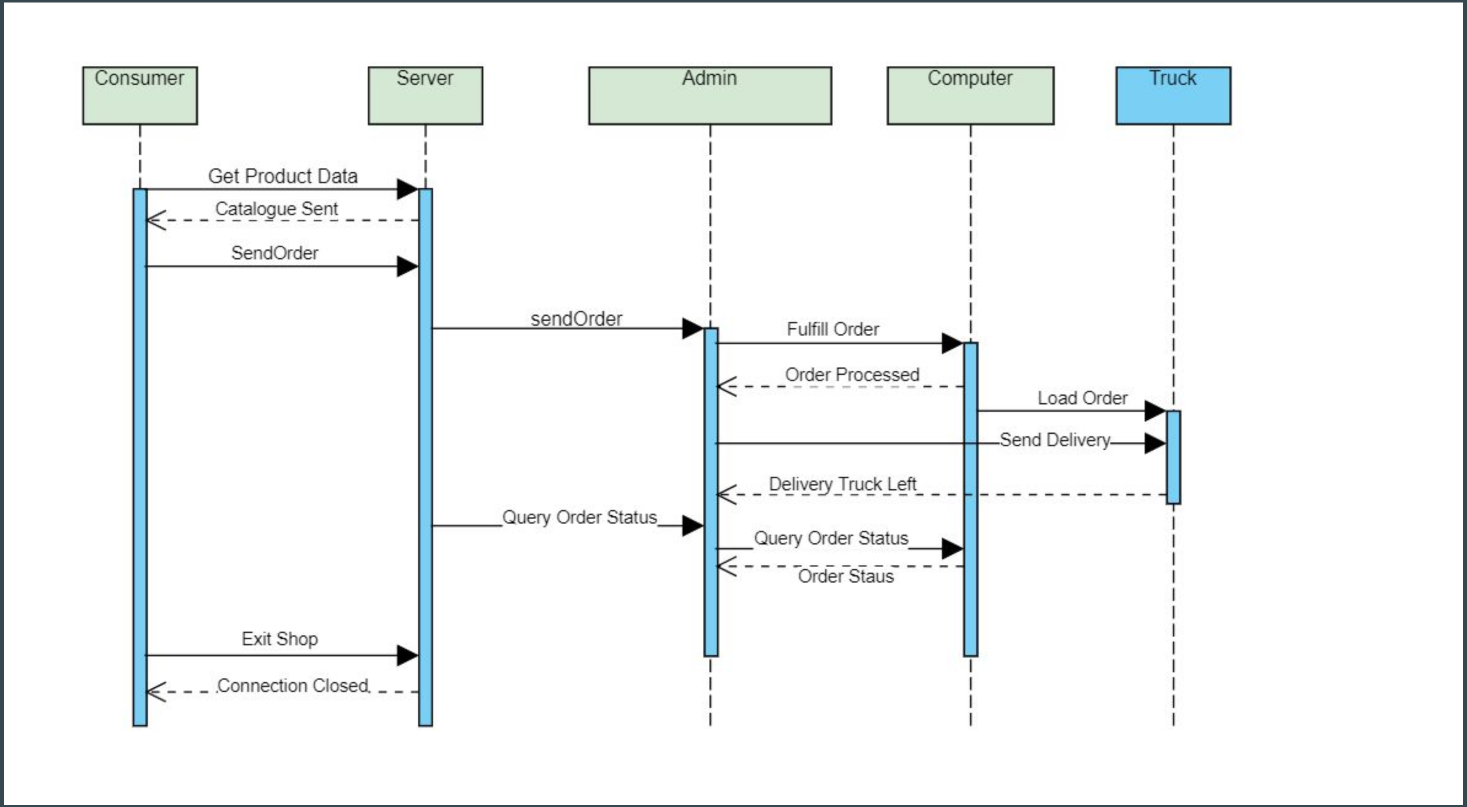
Use Case Diagram:



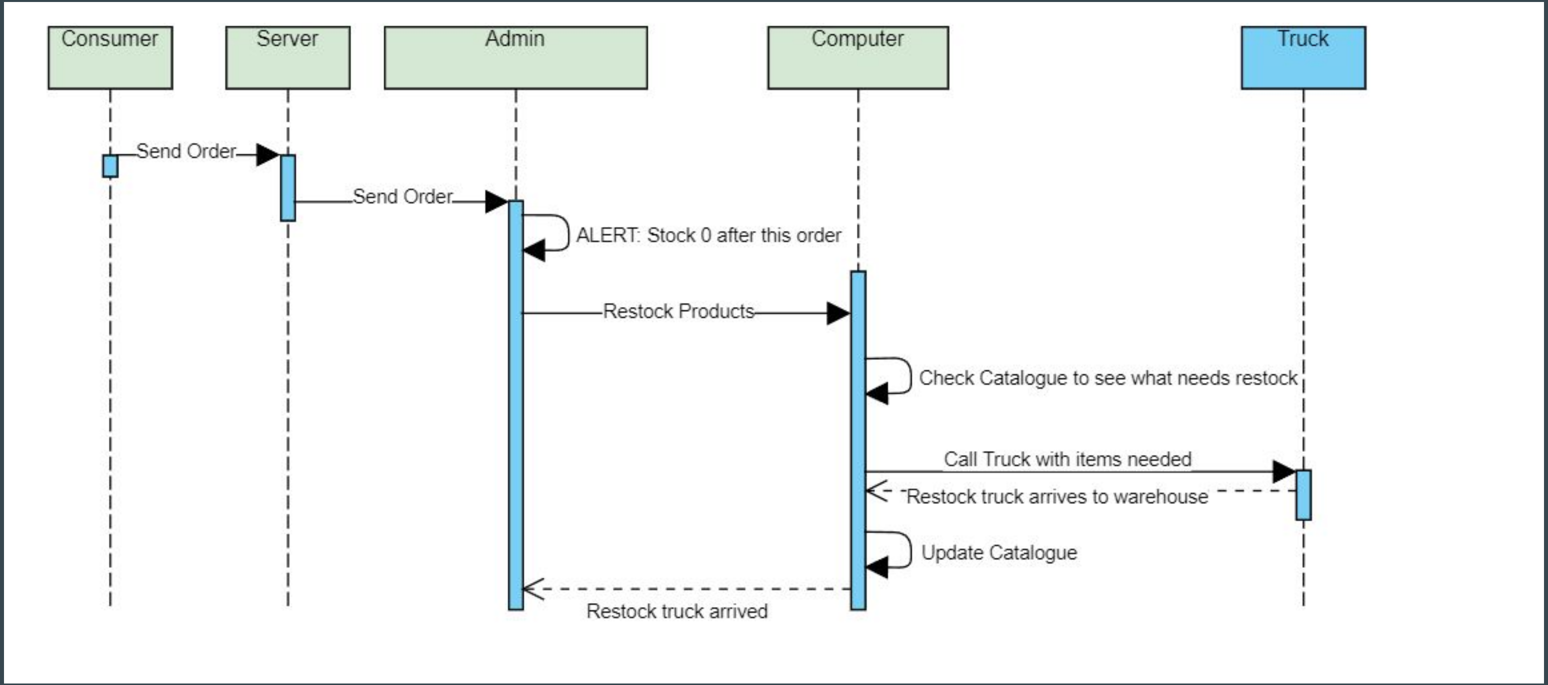
Class Diagram:


Computer (Warehouse)
+shelves:Shelf[]
+workingRobots:List<Robot>
+standbyRobots: Queue<Robot>
+trucks:Truck[]
+maxTruckCapacity:double
+maxItemStock:int
+processedOrders:Queue<Order>
+orderBin:List<Order>
+orderLog:List<(Order,int)>
+dockingQueue:Queue<Truck>
+restockTruckQueue:Queue<RestockTruck>
+deliveryTruckQueue:Queue<DeliveryTruck>
+fulfillOrder(order:Order):void
+OrderIsValid(order:Order):bool
+loadProcessedOrders(currTruck:DeliveryTruck):void
+serviceNextTruck(newNewTruck:bool):DeliveryTruck
+deliverOrders(DeliveryTruck truck):void
+ReadAndReplaceCatalog():string
+AddNewCatalogItem(item:Product):void
+discontinueProduct(product:Product):void
+queryOrderStatus(order:Order):string
Robot
-id:int
-currBatteryLevel:double
-maxLoadingCap:double
-currentLoad: double
-location:int[]
-isActive: bool
robotQueue:Queue<(Item,Shelf)>
+getOrder(gridCellMutices:Mutex[,]):void
-rechargeBattery():void
+QueueItem(item:(Item,Shelf, quantity:int)
1..*
has
1
1
receives
0..*
Truck
+id:int
+maxWeightCapacity:double
+currWeight:double
1
has
0..*
1
Contains
0..*
1
has
1..*
Order
+id: int
+products:List<(Product,int)>
+status:string
1..*
1..*
BasicItem
+name:string
+weight:double
+price:double
+id:int
Shelf
+items:List<Item>
+shelfLocation: ShelfLocation
+id:int
+currWeight:double
+maxWeight:double
1
has
0..*
Product
+stock:int
Item
+shelfId:int
RestockTruck
+items:List<(Product,int)>
DeliveryTruck
+orders:List<Order>

Place Order Sequence Diagram:

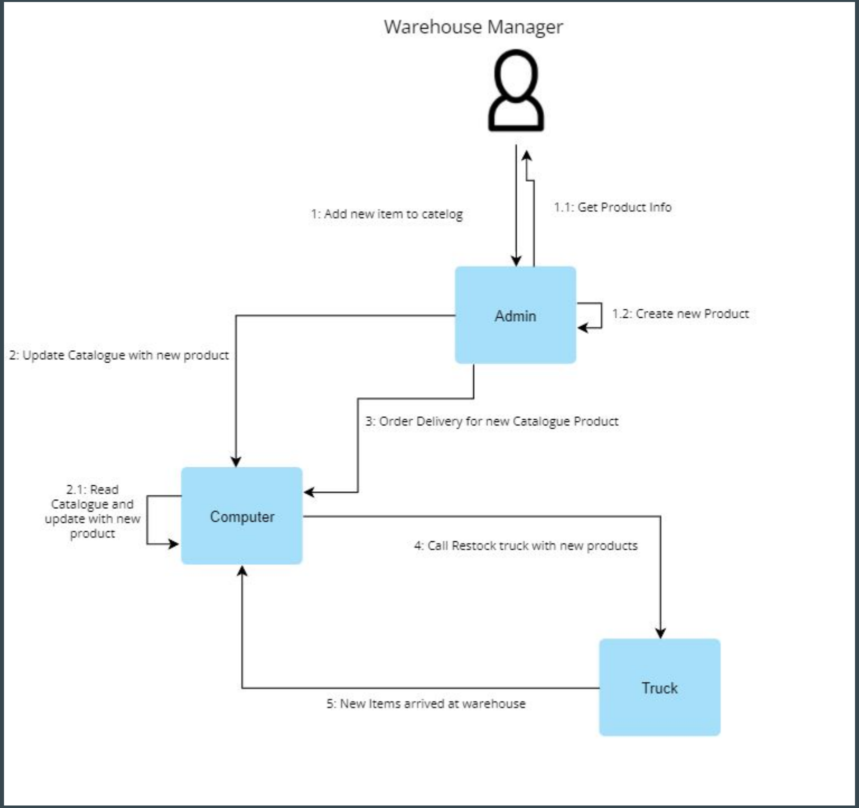


Restock Sequence Diagram:

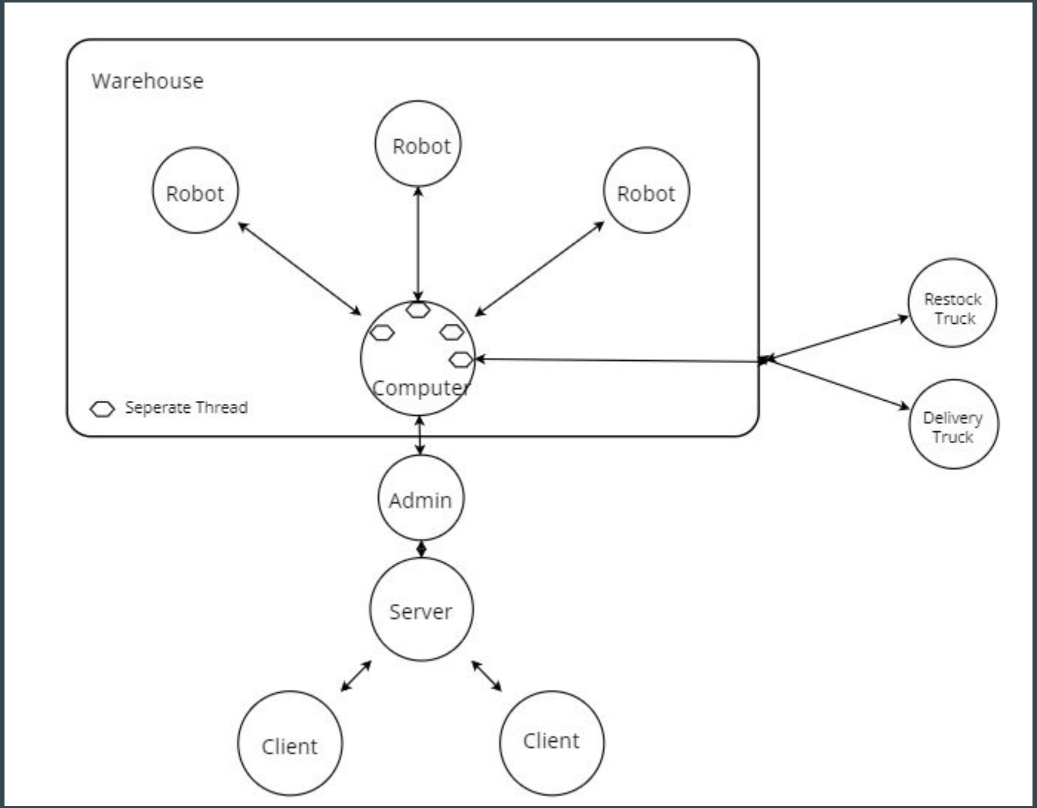


Consumer
Server
Admin
Computer
Truck
Send Order
Send Order
ALERT: Stock 0 after this order
Restock Products
Check Catalogue to see what needs restock
Call Truck with items needed
Restock truck arrives to warehouse
Update Catalogue
Restock truck arrived

Communication Diagram:


Warehouse Manager
1: Add new item to catelog
1.1: Get Product Info
Admin
1.2: Create new Product
2: Update Catalogue with new product
3: Order Delivery for new Catalogue Product
2.1: Read Catalogue and update with new product
Computer
4: Call Restock truck with new products
Truck
5: New Items arrived at warehouse

Object Orientation Diagram:


Warehouse
Robot
Robot
Robot
Computer
Seperate Thread
Restock Truck
Delivery Truck
Admin
Server
Client
Client