Automated Warehouse Proposal

By

Alex Lassooij,

Chaewoon Song,

Nusair Islam,

Phi Lam,

December 8, 2021

CPEN 333

Executive Summary:

Amazoom is the largest internet retailer and one of the fastest-growing companies around the world. They sell a wide range of products and get over 60,000 orders an hour and ship over 1.6 million packages a day.

The Problem:

To effectively and efficiently process large amounts of orders, we propose an automated customizable warehouse that can utilize robots to automatically respond to customer's orders, prepare the items for shipping, and restock items.

Our Solution:

We started off our solution by inquiring the administrator on the layout of the warehouse(rows, columns, and shelves). After the warehouse layout is made in the simulation, the system will constantly monitor item stock, active orders, truck location, and truck weight. A website allows Amazoom customers to order products with plus and minus buttons and add their name to the order. The website updates inventory in real-time and it does not allow customers to order more than the available stock. After an order is placed, the system tracks all active orders and controls the robots to stock the shipping trucks with the items, send the shipping trucks for delivery if full, and send a restocking truck for anything that's low in stock.

Tech Stack:

Figure 1: Tech stack we chose to build our application

We chose to use the ASP.NET as the framework for our project as it allows us to easily build a full-stack web application with a backend written in C# and various options for the frontend. We chose an MVC web application since we thought our website would not be too complex and could be implemented with some simple CSHTML and CSS styling. Later on in the project, we made use of Javascript to make the web pages dynamic. The use of MVC enabled us to implement routing and create multiple web pages with data pulled from our databases. We used Winform to create the GUI as we found it simpler to use for our administrator to input flexible warehouse layouts, see order statuses, see the robots move in the warehouse, see the inventory, and get warnings about low stock items.

For our database, we chose to go with MongoDB, a NoSQL database because it allowed us to create nested tables. We found it hard to nest tables (e.g. a list of items belonging to an order) as we tried with an MS SQL database. With the database running, both the central computer and web server were able to perform CRUD operations on it.

Class Diagram:

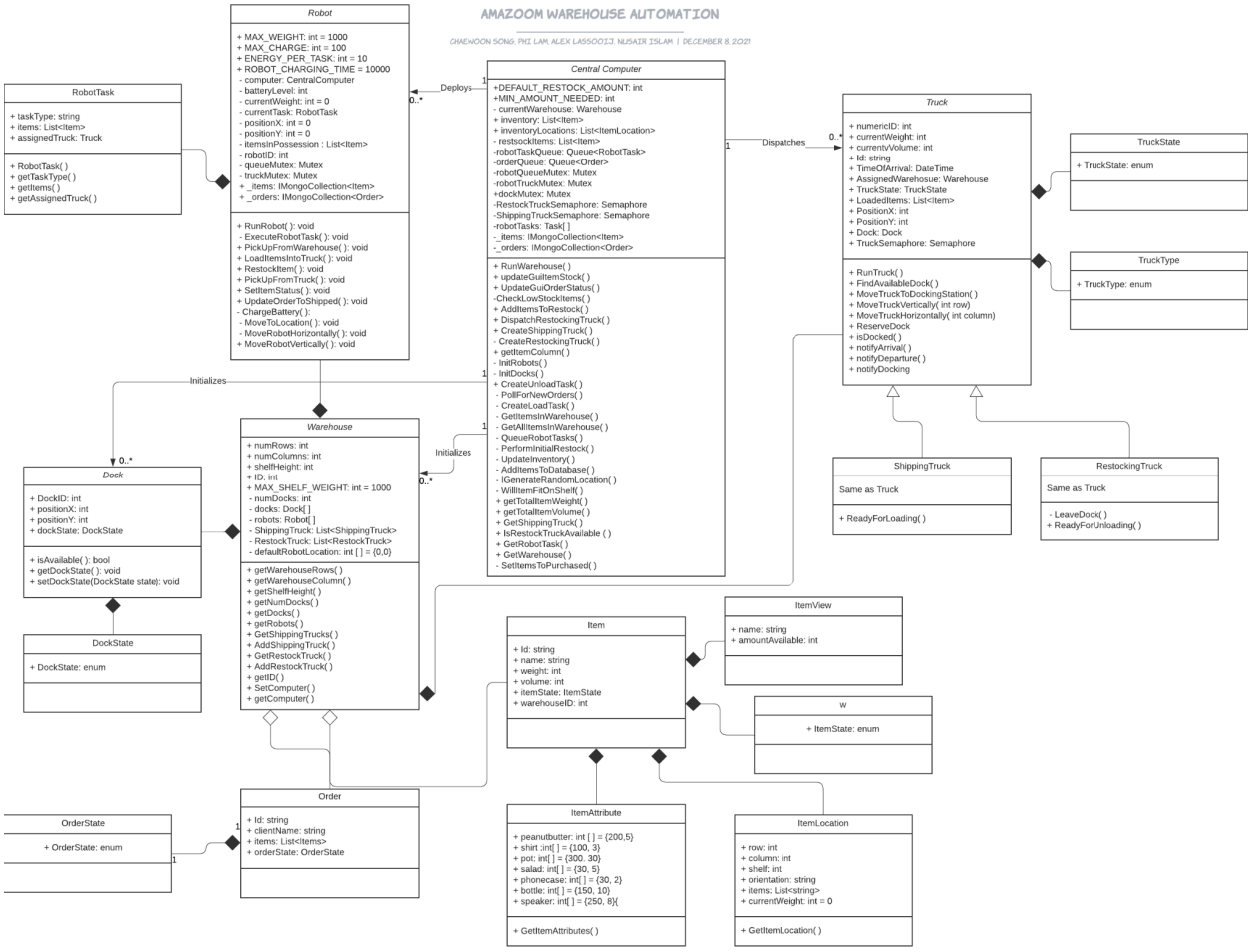

Figure 2: Class Diagram of the system

Use Case Diagram:

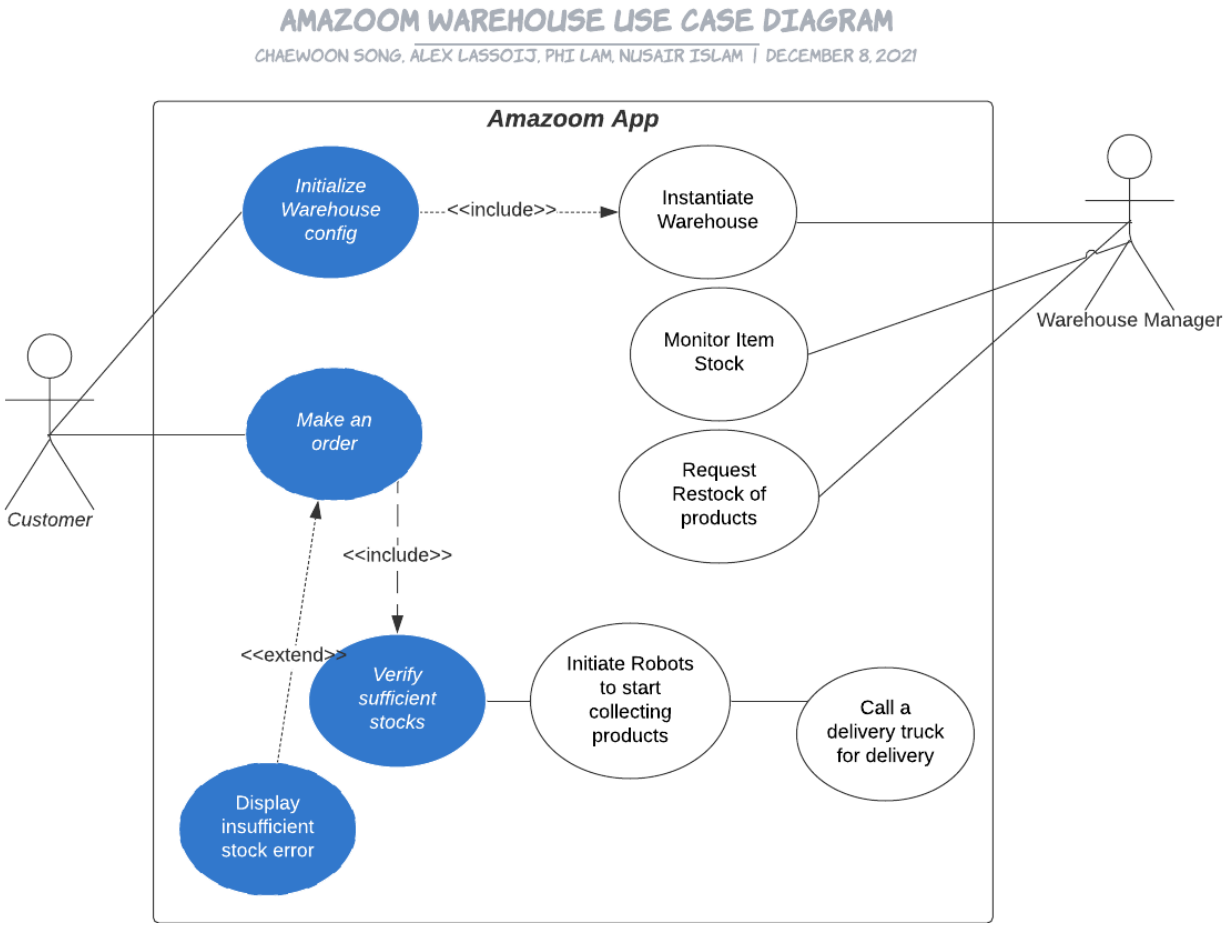

Figure 3: Use Case Diagram of the system

Sequence Diagrams:

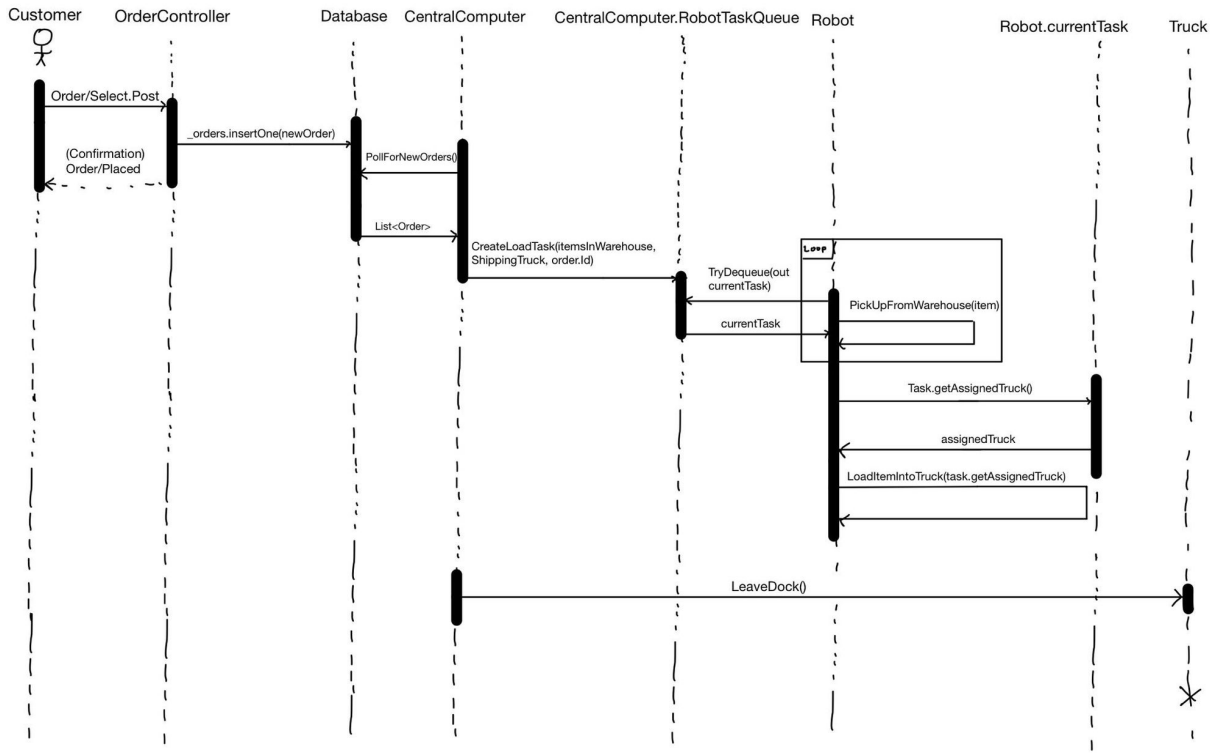

Figure 4: Customer ordering an item sequence diagram

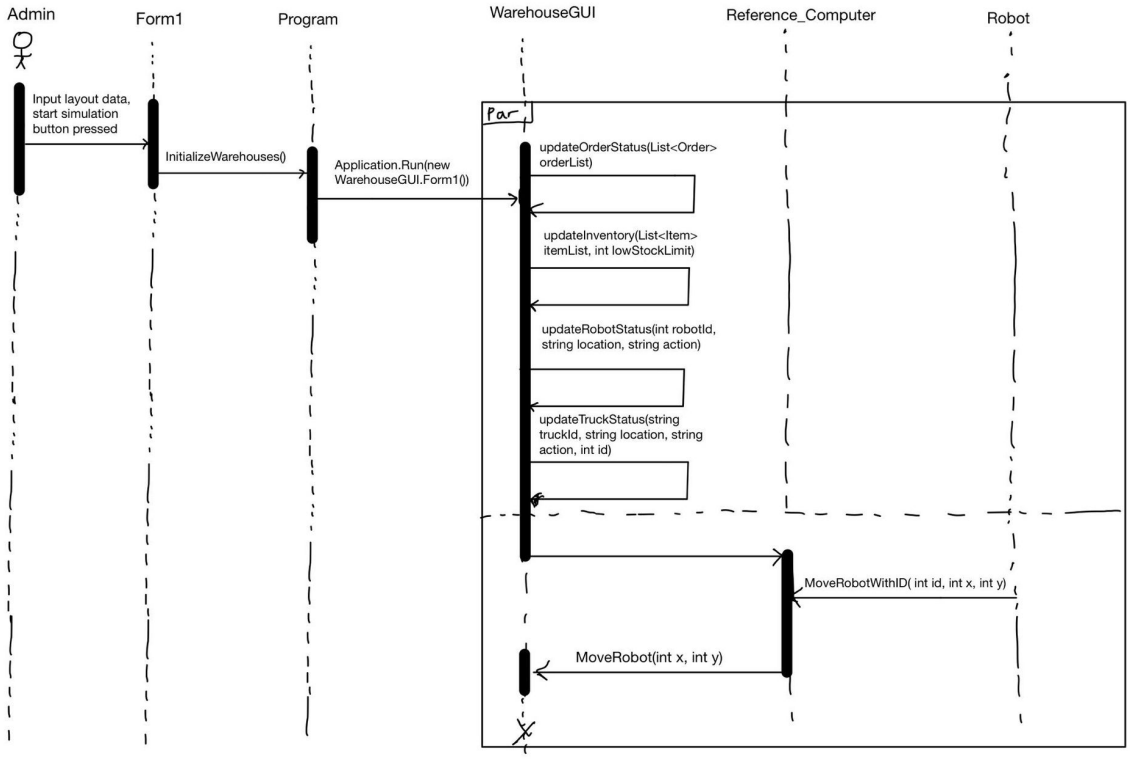

Figure 5: Administrator setting inputting warehouse layout sequence diagram

Communication protocols:

In our warehouse system, communication between the Central Computer and robots and trucks is facilitated through task queues and states mapped to enum values.

In an infinite while loop, the computer polls (with a set timeout) for new orders, whether any items are low in stock and checks the statuses of any trucks.

When a new order is placed by a customer or an incoming truck with new items docks at the warehouse, the central computer reacts by enqueueing tasks in a *RobotTaskQueue*, which is a queue with elements of type RobotTask. Each RobotTask object specifies whether the robot's task is to load/unload a truck, what truck it will load to / unload from and the list of items it should load / unload. Each RobotTask is designed to be consumed by exactly one robot. The list's items are ensured to not exceed the robot's maximum weight capacity. The dequeuing of tasks is also mutually exclusive. In essence, the control of robots by the computer is implemented through a producer - consumer pattern.

However, the central computer only enqueues new tasks under certain set of conditions. For example, if a new order is detected, the central computer will only enqueue loading task(s) if an available shipping truck is docked that can fit the additional weight and volume of the new

order. To check whether a truck is docked, the computer will look at the truck's status, whose values are any of the values in *TruckState*.

We also added states for items and orders in *ItemState* and *OrderState*, respectively. Upon reaching different stages in the shipping process, items will have their state updated accordingly both in the warehouse's inventory and the database. For example, an item's state will be *Purchased* right after the customer orders it. When it is loaded into a truck, its state will be *Loaded*. The computer has the ability to check the states of a particular order's items. If all items have the state *Loaded* and the truck they were loaded in has departed, the order's state will switch to *Shipped*.

Before attempting to dock at the warehouse, any truck will check if a dock is in the *Available* DockState. We also added a semaphore, where the max entries are equal to #Docks / 2 + # Dock % 2. This avoids the case where all docks could be occupied by shipping trucks. This could lead to a problem, since shipping trucks only leave the warehouse once their total weight exceeds a set value. If this happens, no restocking trucks will be able to arrive until more orders are loaded into the truck. In the worst case scenario, the warehouse might completely run out of items to load, leading to a deadlock, where shipping trucks are waiting to have more weight added, and restocking trucks are waiting to dock. With the use of this semaphore, this situation is avoided.

Function specifications:

PollForNewOrders in CentralComputer.cs :

- Fetches a list of orders that have not been handled yet (i.e. computer has not enqueued tasks for robots to fulfill this order yet)
- Computer checks which items within the order are inside the current warehouse (applicable to a network of warehouses)
- Checks if a shipping truck is available to carry the items belonging to the order
- If a shipping truck is found but cannot carry the additional weight, the truck is ordered to depart the warehouse and a new shipping truck is requested
- Queues task(s) for robots to load the order's items into the truck found
- Updates the status of the items and order to *Loading*

CheckLowStockItems() in CentralComputer.cs :

- Fetches the amount of each item in the warehouse and adds data to dictionary, with the item name as the key and the amount as the value
- If any items' amounts are below *MIN_AMOUNT_NEEDED*, the computer will add *DEFAULT_RESTOCK_AMOUNT* x items of this type to a list called *restockItems*. *restockItems* is a list of items that will be brought into the warehouse by a new restocking truck
- Once the total weight of all items in *restockItems* exceeds *MAX_WEIGHT* in Truck.cs, a new restocking truck is dispatched with the items contained in *restockItems*, and *restockItems* is cleared.
- The computer then enqueues tasks for robots to unload the incoming restocking truck.