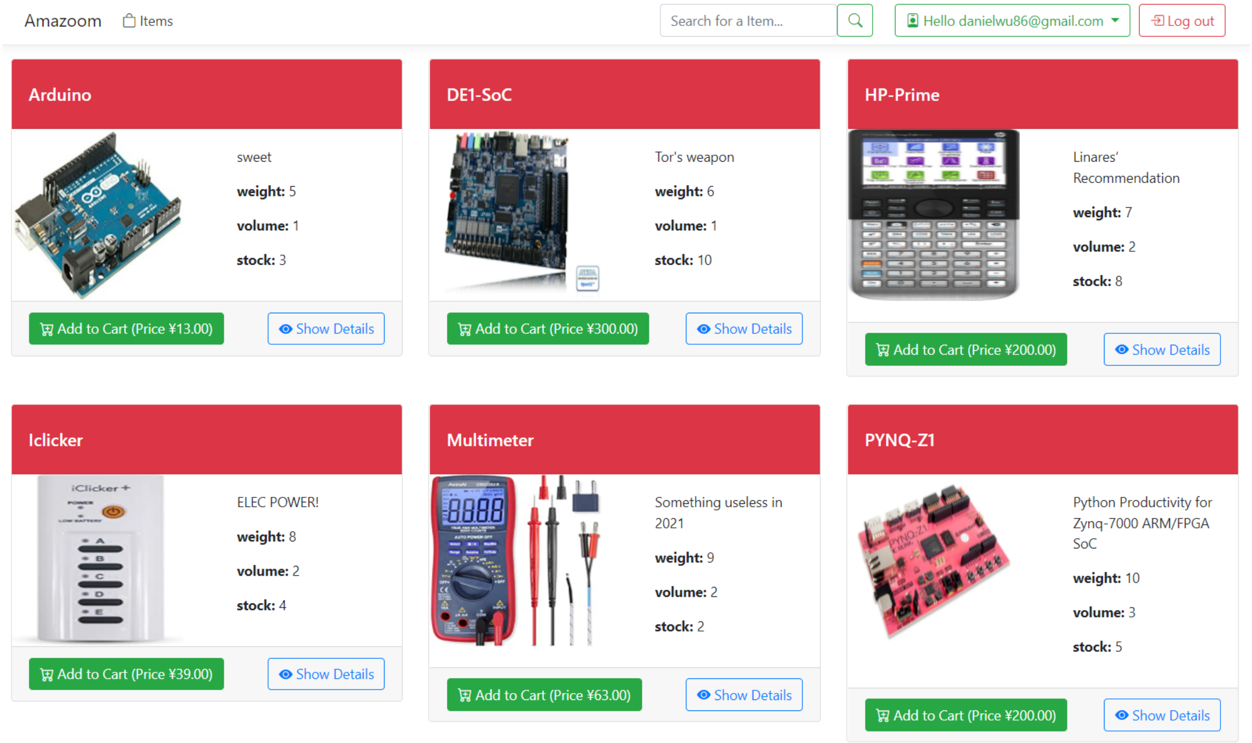
Project Amazoom

System Design Document

(2021W CPEN333)

08 December 2021


Amazoom
Items
Search for a Item...
Hello danielwu86@gmail.com
Log out

Arduino

sweet

weight: 5

volume: 1

stock: 3

Add to Cart (Price ¥13.00)
Show Details

DE1-SoC

Tor's weapon

weight: 6

volume: 1

stock: 10

Add to Cart (Price ¥300.00)
Show Details

HP-Prime

Linares' Recommendation

weight: 7

volume: 2

stock: 8

Add to Cart (Price ¥200.00)
Show Details

Iclicker

ELEC POWER!

weight: 8

volume: 2

stock: 4

Add to Cart (Price ¥39.00)
Show Details

Multimeter

Something useless in 2021

weight: 9

volume: 2

stock: 2

Add to Cart (Price ¥63.00)
Show Details

PYNQ-Z1

Python Productivity for Zynq-7000 ARM/FPGA SoC

weight: 10

volume: 3

stock: 5

Add to Cart (Price ¥200.00)
Show Details

Project Members

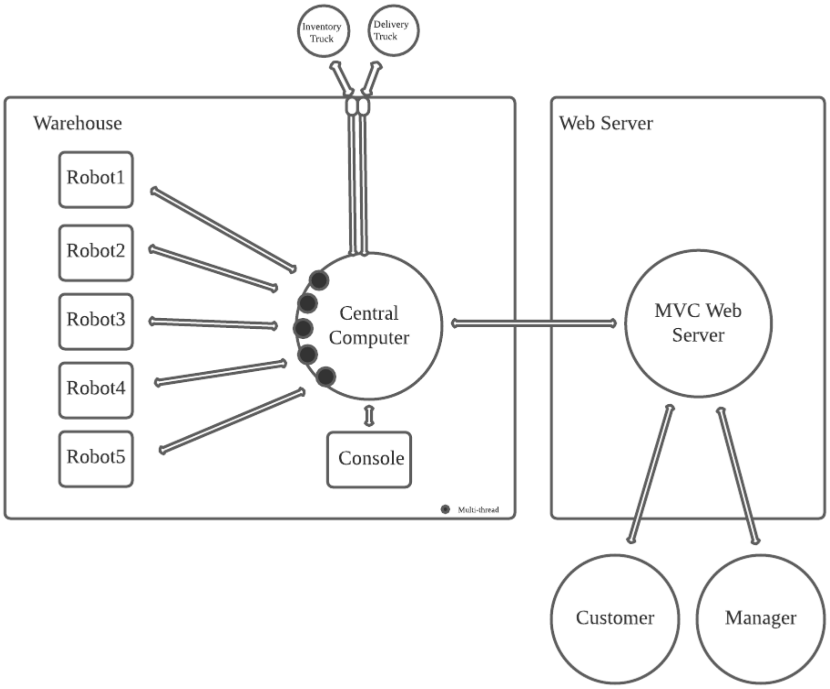
Bolong Tan 52513280
Chen Qin 82138900
Tianyu Wu: 80350507
Trent Diao: 28319556

Table of Contents

1 Executive Summary

A warehouse simulation has been created in order to keep track of operations from a remote location. This warehouse is capable of automatically transferring items based on the actions of online users through the use of autonomous robots. Each robot runs our custom-built algorithm to be able to detect new orders, transfer items, and navigate the warehouse without collisions. All of this is handled by the warehouse central computer, which the manager can connect to using the website. In addition, the warehouse manager is able to see in real-time the movement of the robots and order items to restock the warehouse whenever they choose.

The website has a user-friendly interface that contains client-based functions such as placing orders, a search bar, order records, and item details. The manager can change all the item info such as ID, name, description, and stock (connected to the database and warehouse). The registration system authorizes clients and managers access to different functions. The server allows multiple clients to place orders. The customer data and item data are stored in a SQL database for safety. The ordering and restocking system can change the database automatically in real-time. The ordering and restocking functions of the website are connected to the warehouse using the TCP socket, sending strings and decoding the string into actual instructions.



2 Tech Stack

2.1 Warehouse Side

A console application project is used for the warehouse project. The whole warehouse operation is coded with C#, an object-oriented programming language used to develop and integrate with the .NET platform.

2.2 Webserver Side

The ASP.NET MVC (Model, View, Controller) is used as the basic framework for the webserver. It enables complete control over the rendered HTML based on the created Models and Views. The separation between backend software logic and frontend view provides a better division of labor and improved maintenance during development. The SQL database is used to store user information, instant shopping cart item information, admin management information, etc. It provides an easy way to store large numbers of records efficiently and is easy to edit simultaneously.

2.3 Warehouse-Webserver Connection

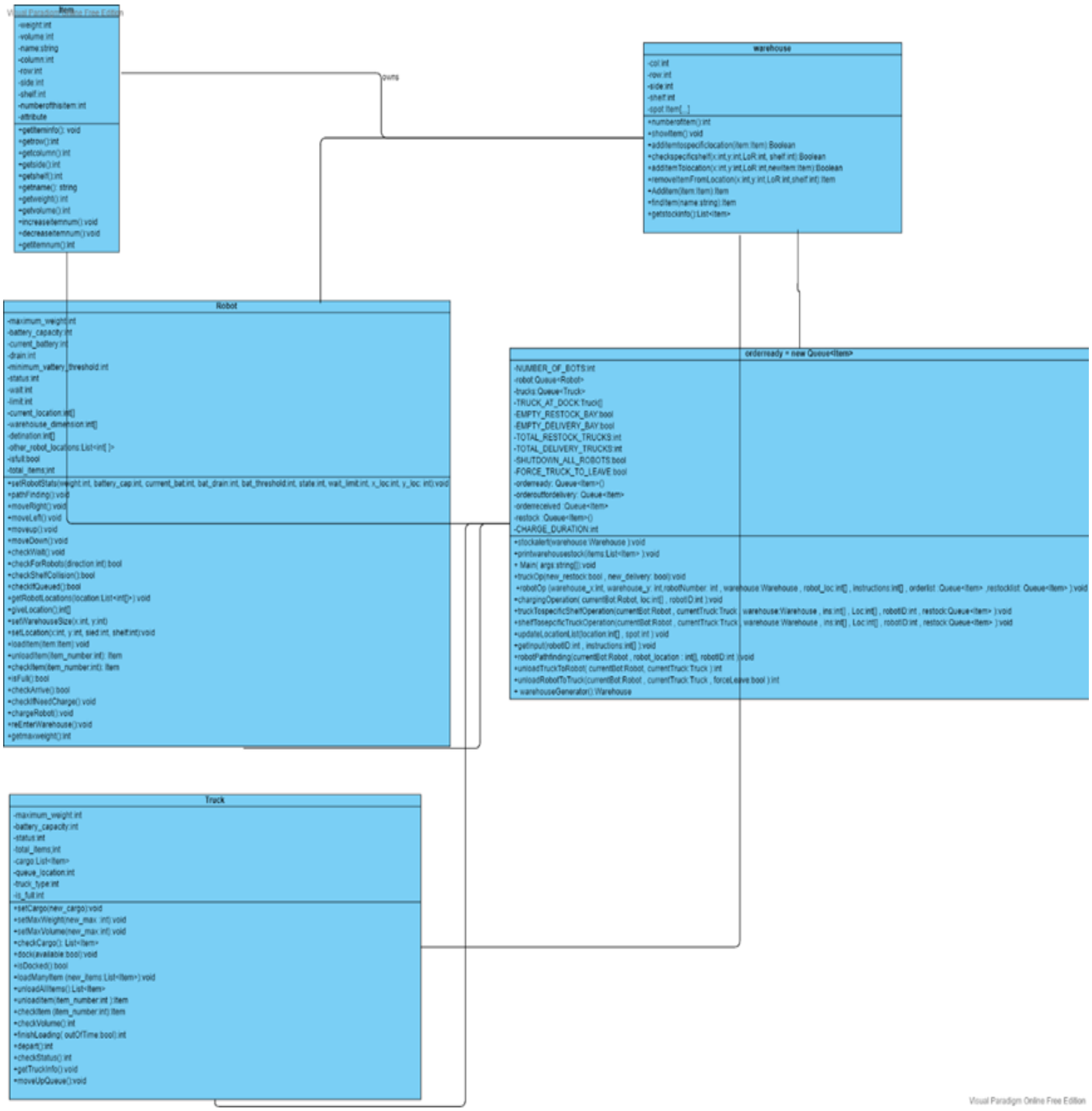
The TCP socket is used for data/command transmission between the warehouse and the webpage server. It provides sufficient transmission performance while improving the coding efficiency and reducing project learning costs.

3 System Design

3.1 Warehouse System Design

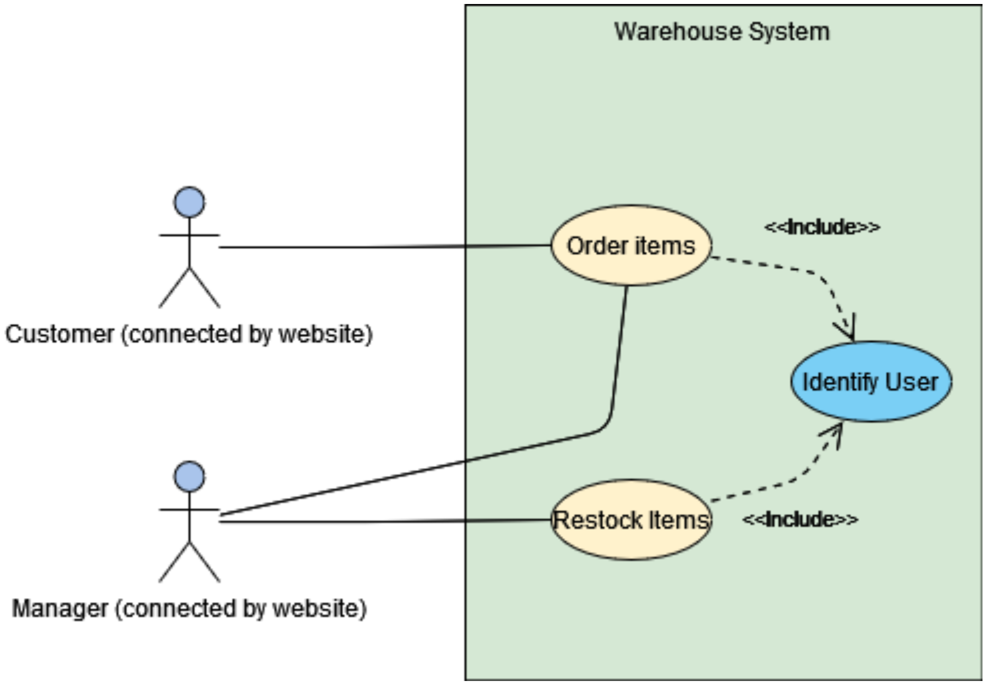
3.1.1 Class Diagram

The class diagram of the backend includes central computer class, robot class, item class, truck class, and warehouse class. The relationship between classes is shown below.



Visual Paradigm Online Free Edition
Item
-weight:int
-volume:int
-name:string
-column:int
-row:int
-side:int
-shelf:int
-numberofthisitem:int
-attribute
+getiteminfo(): void
+getrow():int
+getcolumn():int
+getside():int
+getshelf():int
+getname(): string
+getweight():int
+getvolume():int
+increaseitemnum():void
+decreaseitemnum():void
+getitemnum():int
owns
warehouse
-col:int
-row:int
-side:int
-shelf:int
-spot:Item[...]
+numberofitem():int
+showitem():void
+additemtospecificlocation(item:Item):Boolean
+checkspecificshelf(x:int,y:int,LoR:int, shelf:int):Boolean
+additemTolocation(x:int,y:int,LoR:int,newitem:Item):Boolean
+removeitemFromLocation(x:int,y:int,LoR:int,shelf:int):Item
+Additem(item:Item):Item
+finditem(name:string):Item
+getstockinfo():List<Item>
Robot
-maximum_weight:int
-battery_capacity:int
-current_battery:int
-drain:int
-minimum_vattery_threshold:int
-status:int
-wait:int
-limit:int
-current_location:int[]
-warehouse_dimension:int[]
-detination:int[]
-other_robot_locations:List<int[ ]>
-isfull:bool
-total_items:int
+setRobotStats(weight:int, battery_cap:int, current_bat:int, bat_drain:int, bat_threshold:int, state:int, wait_limit:int, x_loc:int, y_loc: int):void
+pathFinding():void
+moveRight():void
+moveLeft():void
+moveup():void
+moveDown():void
+checkWait():void
+checkForRobots(direction:int):bool
+checkShelfCollision():bool
+checkifQueued():bool
+getRobotLocations(location:List<int[]>):void
+giveLocation():int[]
+setWarehouseSize(x:int, y:int)
+setLocation(x:int, y:int, sied:int, shelf:int):void
+loadItem(item:Item):void
+unloadItem(item_number:int): Item
+checkItem(item_number:int): Item
+isFull():bool
+checkArrive():bool
+checkIfNeedCharge():void
+chargeRobot():void
+reEnterWarehouse():void
+getmaxweight():int
orderready = new Queue<Item>
-NUMBER_OF_BOTS:int
-robot:Queue<Robot>
-trucks:Queue<Truck>
-TRUCK_AT_DOCK:Truck[]
-EMPTY_RESTOCK_BAY:bool
-EMPTY_DELIVERY_BAY:bool
-TOTAL_RESTOCK_TRUCKS:int
-TOTAL_DELIVERY_TRUCKS:int
-SHUTDOWN_ALL_ROBOTS:bool
-FORCE_TRUCK_TO_LEAVE:bool
-orderready: Queue<Item>()
-orderoutfordelivery: Queue<Item>
-orderreceived :Queue<Item>
-restock :Queue<Item>()
-CHARGE_DURATION:int
+stockalert(warehouse:Warehouse):void
+printwarehousestock(items:List<Item>):void
+ Main(args:string[]):void
+truckOp(new_restock:bool , new_delivery: bool):void
+robotOp (warehouse_x:int, warehouse_y: int,robotNumber: int , warehouse:Warehouse , robot_loc:int[] , instructions:int[] , orderlist :Queue<Item> ,restocklist: Queue<Item>):void
+chargingOperation(currentBot:Robot, loc:int[] , robotID:int):void
+truckTospecificShelfOperation(currentBot:Robot , currentTruck:Truck , warehouse:Warehouse , ins:int[] , Loc:int[] , robotID:int , restock:Queue<Item>):void
+shelfTosepcificTruckOperation(currentBot:Robot , currentTruck:Truck , warehouse:Warehouse , ins:int[] , Loc:int[] , robotID:int , restock:Queue<Item>):void
+updateLocationList(location:int[] , spot:int):void
+getInput(robotID:int , instructions:int[]):void
+robotPathfinding(currentBot:Robot , robot_location : int[], robotID:int):void
+unloadTruckToRobot(currentBot:Robot, currentTruck:Truck):int
+unloadRobotToTruck(currentBot:Robot , currentTruck:Truck , forceLeave:bool):int
+ warehouseGenerator():Warehouse
Truck
-maximum_weight:int
-battery_capacity:int
-status:int
-total_items:int
-cargo:List<Item>
-queue_location:int
-truck_type:int
-is_full:int
+setCargo(new_cargo):void
+setMaxWeight(new_max :int):void
+setMaxVolume(new_max:int):void
+checkCargo(): List<Item>
+dock(available:bool):void
+isDocked():bool
+loadManyItem (new_items:List<Item>):void
+unloadAllItems():List<Item>
+unloadItem(item_number:int):Item
+checkItem (item_number:int):Item
+checkVolume():int
+finishLoading(outOfTime:bool):int
+depart():int
+checkStatus():int
+getTruckInfo():void
+moveUpQueue():void
Visual Paradigm Online Free Edition

3.1.2 Warehouse Use-case Diagram



The two main use cases involving the warehouse are ordering items and restocking items. A customer is able to order items, while the manager is able to do both ordering and restocking.

Use Case Scenario: Customer Order

1. Customer opens website
2. <include> identify user by logging in
3. Customer chooses items on the website to order

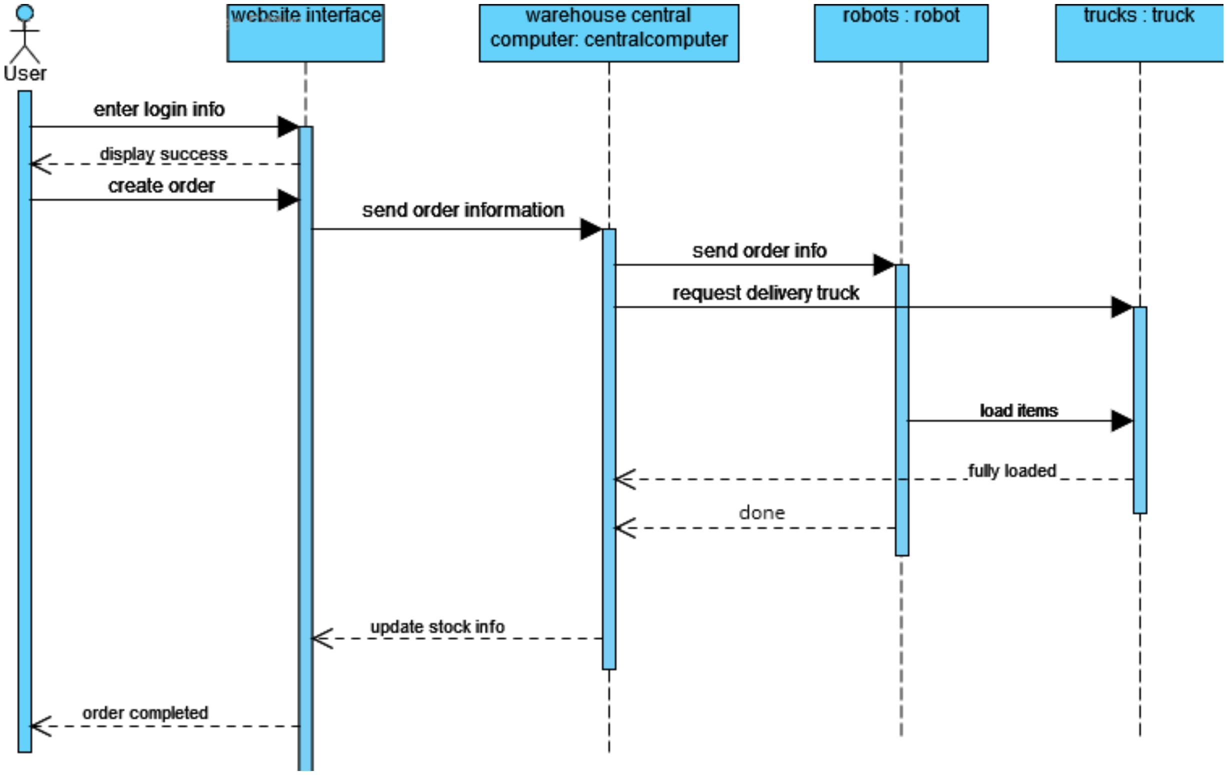
Use Case Scenario: Manager Restock

1. Manager opens website
2. <include> identify the user and verify it is the admin account
3. Customer clicks on restock function and orders items to be restocked

3.1.3 Warehouse Sequence Diagram

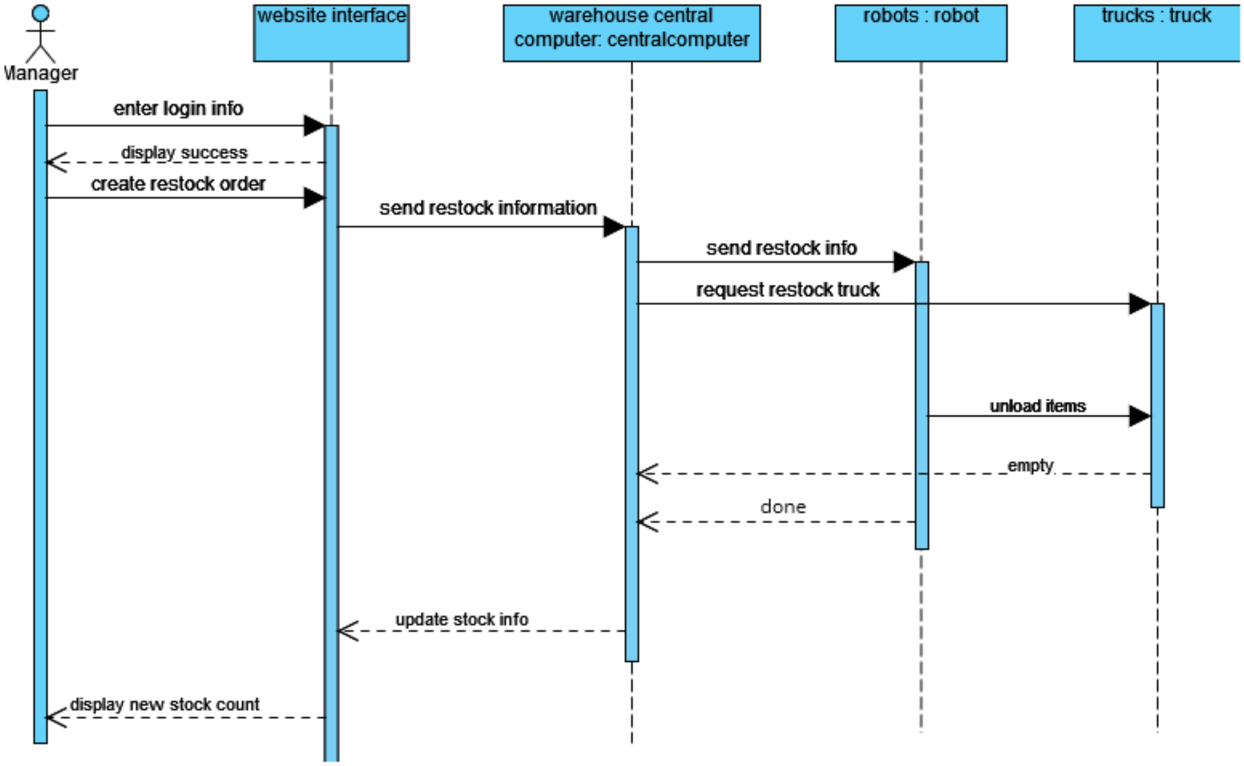
Below are the important sequence diagrams for the warehouse.

Order



When a user makes an order, first they must log in to confirm their credentials. The website will then allow for orders to be made. Once an order has been created, the details are sent to the warehouse central computer. The central computer activates the robots to go retrieve the specified items and requests a new delivery truck to send the items out. After the robots are done loading the truck, the truck leaves and the central computer updates the warehouse stock to reflect the transaction. The website then tells the user that their order has been completed.

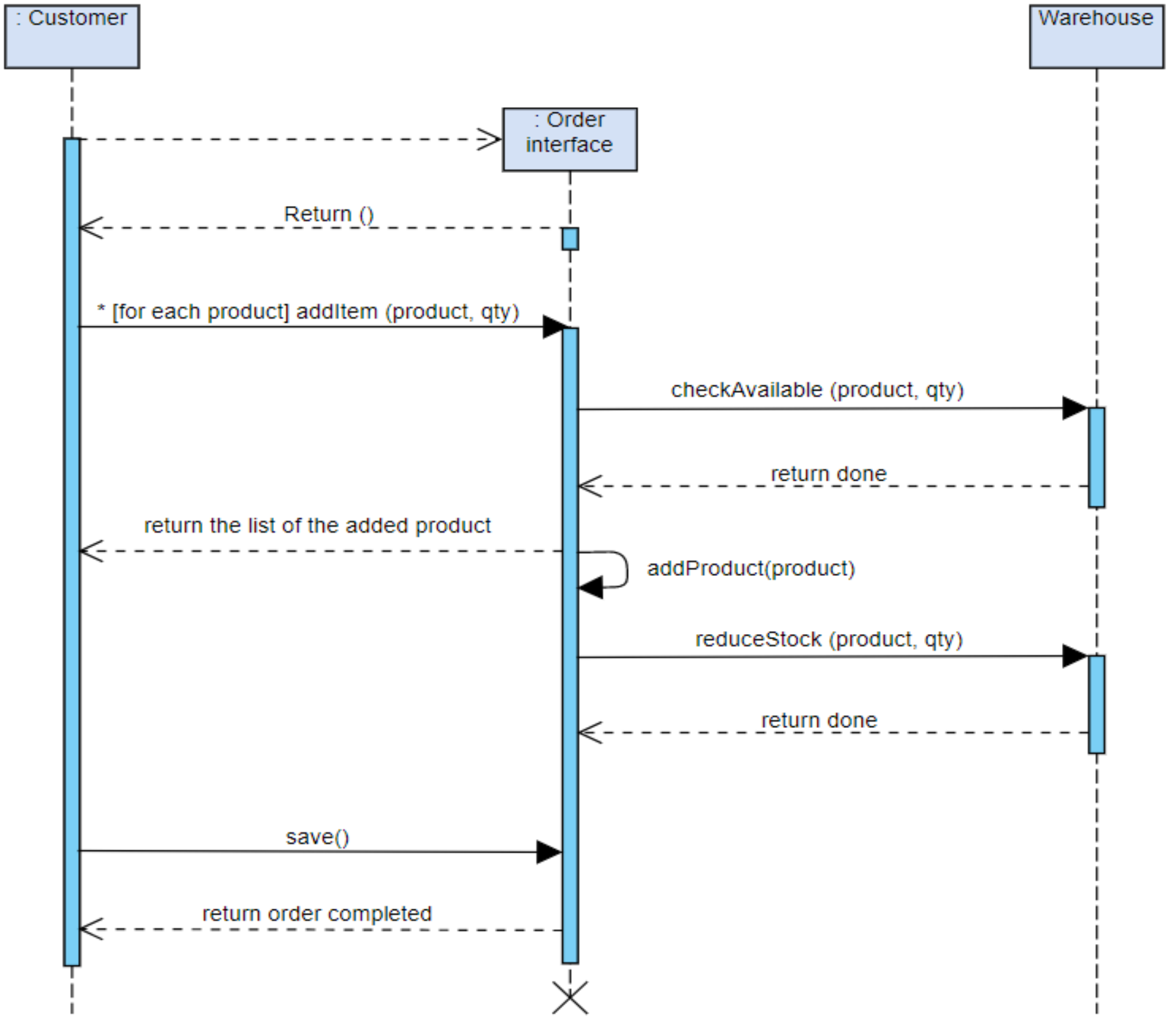
3.1.4 Restock Sequence Diagram



When the manager wants to restock the warehouse, first they must login to confirm their credentials. If they have an admin account, the website will then allow for restock orders to be made. Once the restock order has been created, the details are sent to the warehouse central computer. The central computer activates the robots to go retrieve the specified items from the truck once the new restock truck arrives. After the robots are done unloading the truck, the truck leaves and the central computer updates the warehouse stock to reflect the transaction. The website then shows the manager the new quantity of items in the warehouse.

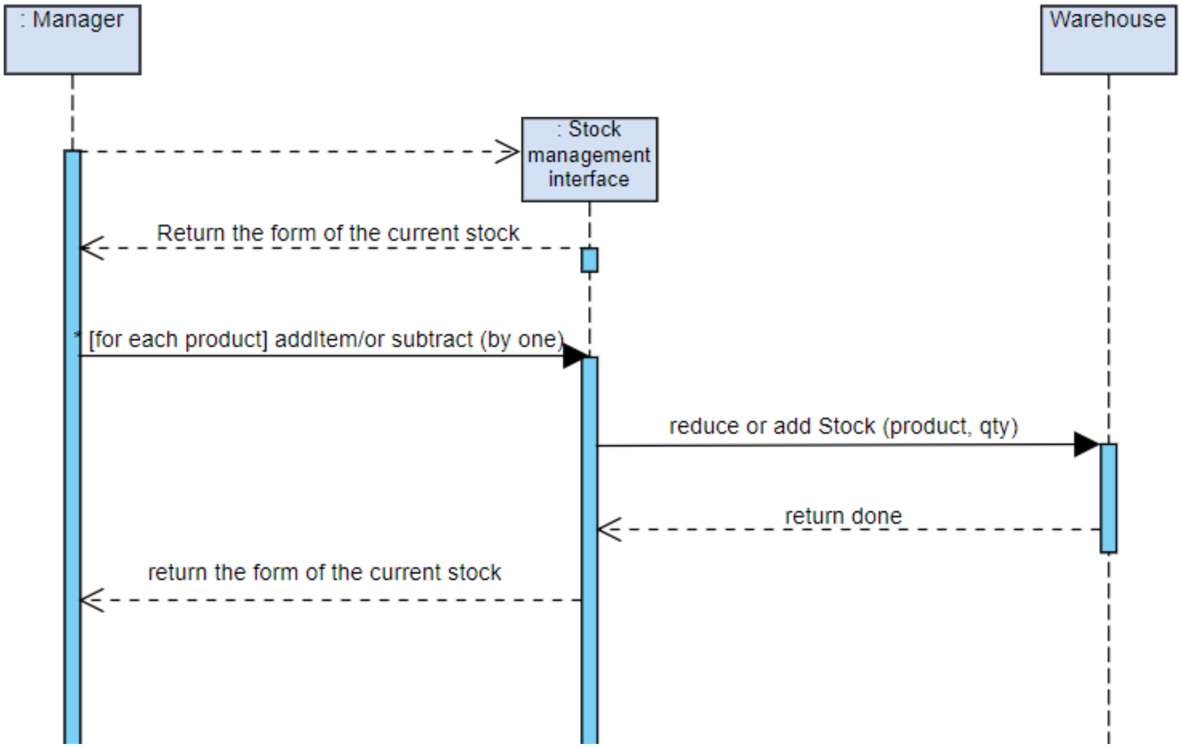
3.2 Web System Design

3.2.1 Client to website Sequence Diagram


: Customer
Warehouse
: Order interface
Return ()
* [for each product] addItem (product, qty)
checkAvailable (product, qty)
return done
return the list of the added product
addProduct(product)
reduceStock (product, qty)
return done
save()
return order completed

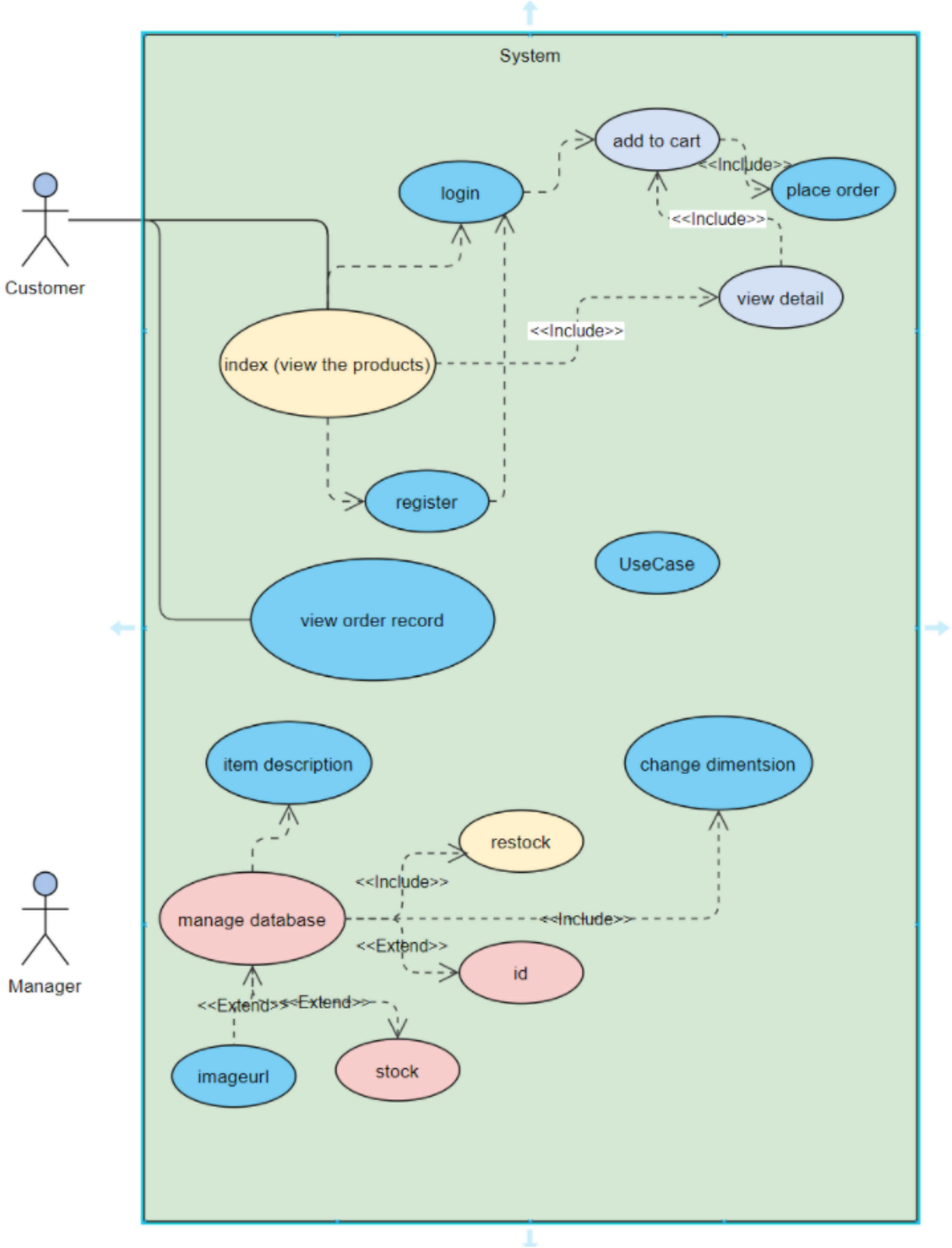
The customer can view the items and place orders. The shopping cart interface changes in real-time while the customer adds the items in real-time. After placing the order, the website sends the instruction to the warehouse.

3.2.2 Manager to Website Sequence Diagram

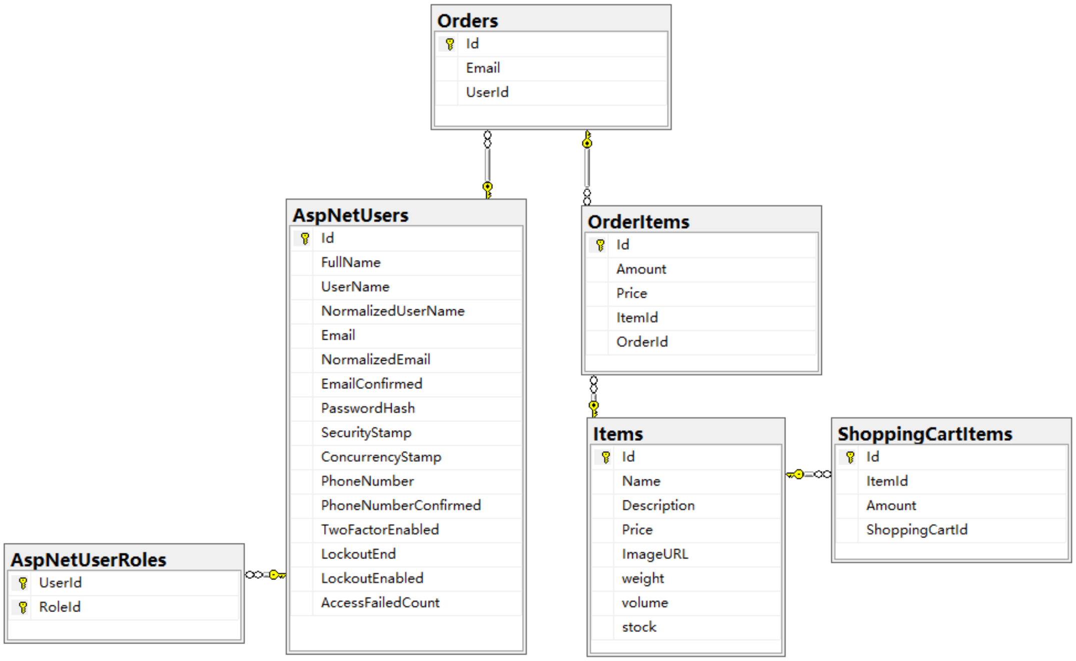

: Manager
Warehouse
: Stock management interface
Return the form of the current stock
* [for each product] addItem/or subtract (by one)
reduce or add Stock (product, qty)
return done
return the form of the current stock

The sequence diagram shows the procedure when the manager is trying to change the stock. The manager views the stock info and on the stock interface. The stock then connects with the warehouse to do the actual operation.

3.2.3 Use Case Diagram



3.2.4 Database Diagram



The database diagram above shows the entity-relationship between different data tables migrated and created from the ASP.NET MVC framework. In the diagram, a KEY indicates a single relationship, and an infinity symbol indicates a "many" relationship. I.e. For the connection between the Items table and the ShoppingCartItems table, there is a key at one end and an infinity symbol on the other. This represents a one-to-many relationship that existed in the database entity.

THE END