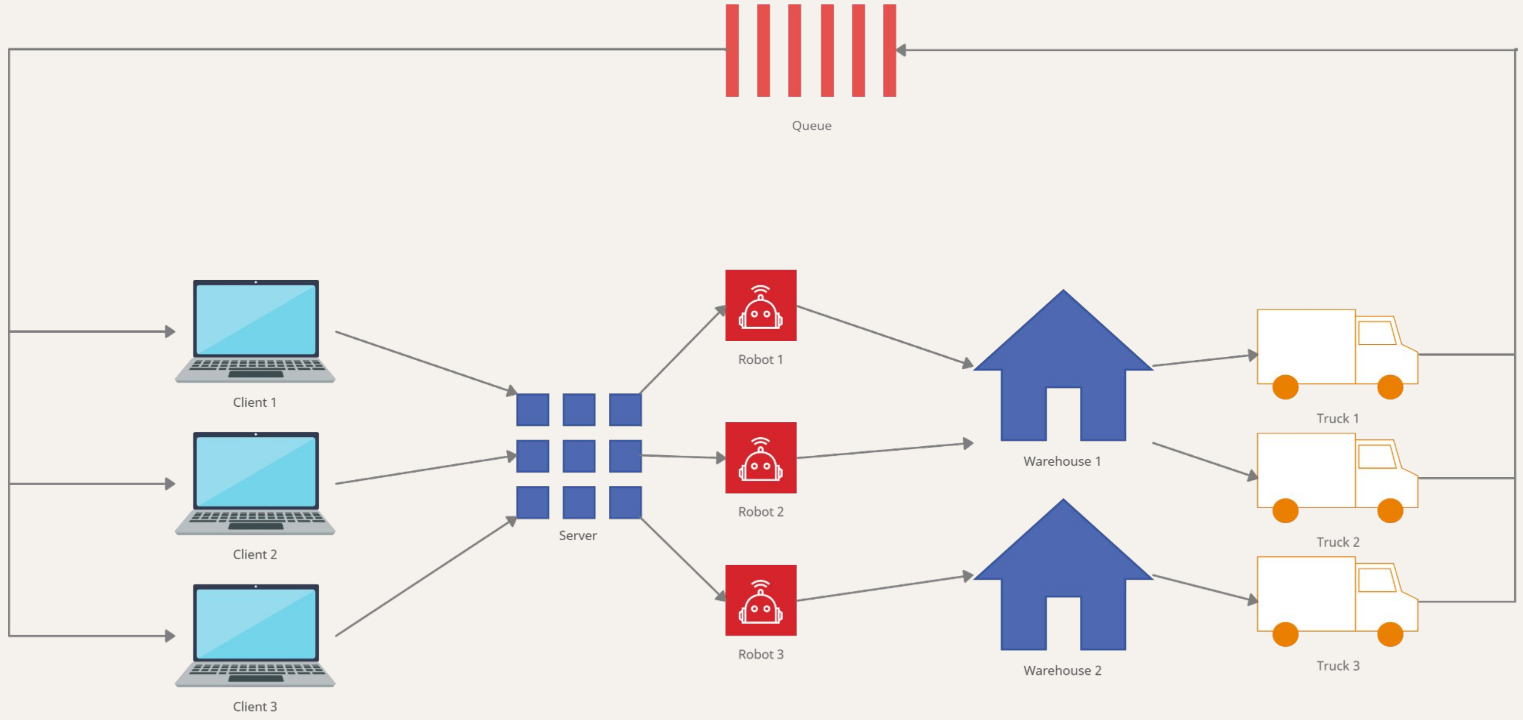
# CPEN 333 Project

CK Chan, Mahmoud Abdelhadi, Jadon Hladyshevsky, Jin Hee Yun

## Tech Stack

- Front End: ASP.NET with MVC model
  - Separates frontend actions from backend
  - While allowing both sides to access its models.
- Back End: C#
  - Required
  - Most of our course material is based on C#

# Overall Design



## Demo

- Show all customer and administrator actions
- View all items in warehouse
- Add to cart and order
- Create flexible warehouse sizes
- Create new robots when needed

## Authorization & Authentication

- We implemented caches to keep users logged in even if they leave the page
- Manager
  - We have added a Manager role which can be added as a new policy if our client would like for managers only to do do certain tasks
    - The manager login is manager@Amazoom.com
- Admin
  - Currently, We have the Administrator role, where we have an authentication policy in place that checks if the user logged in has the admin claim to access any admin pages which require the policy requirement of Admin
    - The Admin login is admin@Amazoom.com

## Authorization & Authentication

- User
  - This is our user, where anyone can make a new login with their email and set password, where they can login and browse our items, add items to cart and order items.
  - The user does not have an admin claim for admin policy, hence they will be denied access if they attempt to browse the admin control pages

## Warehouses

- Flexible layouts
  - Our warehouses are added by admins, where they can choose the number of rows, column, the shelf size, and shelf maximum weight
- Warehouse-specific inventories
  - Each warehouse has its own inventory, when adding a new item, the admin picks where to place the item, where they have to specify item name, weight, warehouse stored and quantity.
- User interface
  - The user only sees the inventory in all warehouses, where items with the same name are grouped together and the user can add to cart any item.

## Warehouses

- Low stock warning
    - Admins get a warning when items are low in stock, where they can order more of said items.

## Orders

- Order placement
    - When a user places an item in cart, the stock is updated and the item is reserved until the user either places order or removes it from cart
    - Once the order is placed, the request is sent to the warehouse where the order is placed, where the robot goes picks up the item and stands in queue for the truck
    - Orders are based on user, hence they will not lose the cart if they leave

## Robots

- Warehouse Map
  - Robots keep track of their (x,y) coordinates, their direction and the location of all the other robots
  - Each warehouse created creates a thread-safe boolean array map, where only 1 robot can be at a certain square, each robot occupies the square it is currently in and releases the square that is behind it depending on direction
- Avoidance system
  - We have implemented a clockwise movement for all robots, where they only move in a clockwise rotation to avoid collision, where a robot will wait for the robot in front of it (each square is a mini semaphore)until done to move in the same direction.

## Robot

- Charging
  - We modelled the Robot charge where each 10 steps the robot takes, it loses 1% of its charge, and once it reaches 0, it goes automatically to charging station (0,0)  to recharge
- Charging station
  - Our charging station is “outside” the warehouse, it also charges in the background (new thread) where it fully charges, then the robot can we used again.
  - Our charging model is every 5 seconds the robot gains 1% of charge.

## Robot

- Queue
  - Each robot knows its place in the robot queue, where it is updates once Enqueue is called, once the robot dequeues the place in queue is null.
- Item within the robot
  - Once the robot picks up an item from the shelve, the robot record what item it has and holds it until passed to the truck
- Robot Task Chaining
  - We used task chaining for each robot, where each item to pick up is chained after the antecedent is successfully completed
- Robot Semaphore
  - We implemented a semaphore so only 10 robots can be in the warehouse at a time, other tasks are added onto existing robots if no robots available

## Truck

- Truck Implementation
    - Every warehouse is created with two trucks that wait at the loading bay location of that given warehouse for items to be received from robots.
- Capacity
    - Once a truck is full of items equalling 75% or more of it’s given capacity, it departs and returns in a simulated time-frame of 5 seconds, giving opportunity for the second truck to collect items in the meantime. The trucks will alternate picking up and delivering items depending on where they are at any given moment.
- Dock Capacity
    - Our dock has a semaphore implemented to only allow as many trucks as there are docks, releasing the semaphore once it is undocked, and using the semaphore once docked.