**Amazoom Design Document**

**CPEN 333**

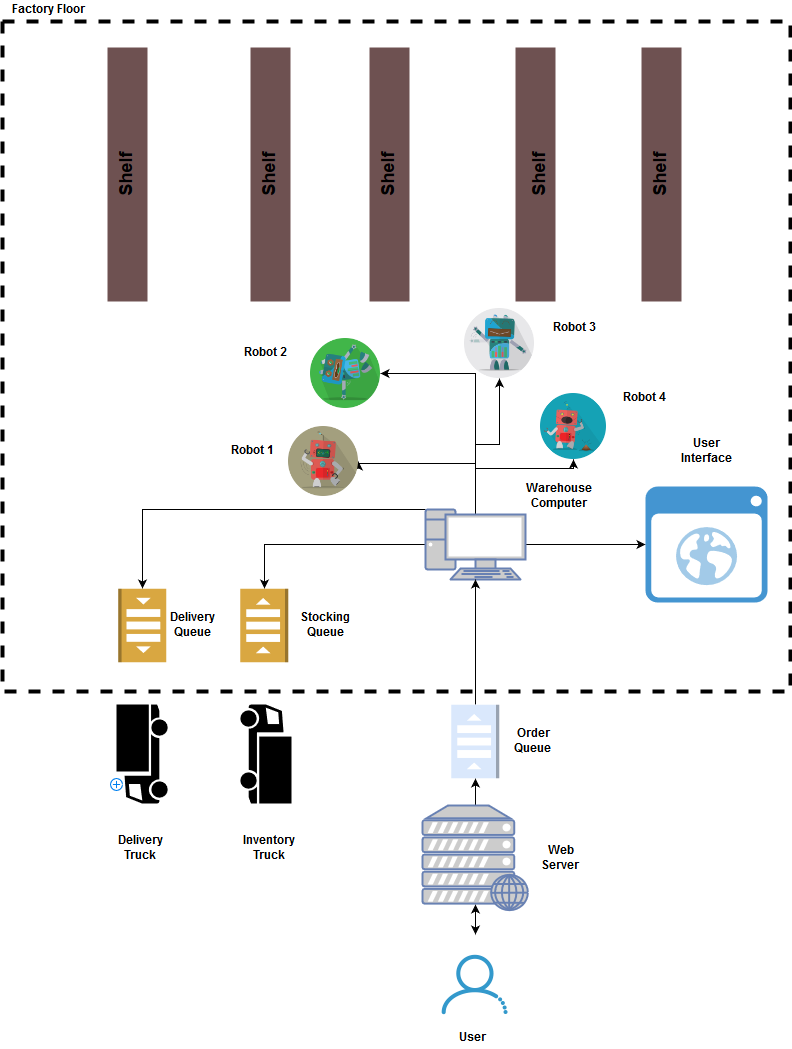
**Moshe Lawlor**

**16868135**

**December 5, 2017**

**Amazoom Automated Warehouse**

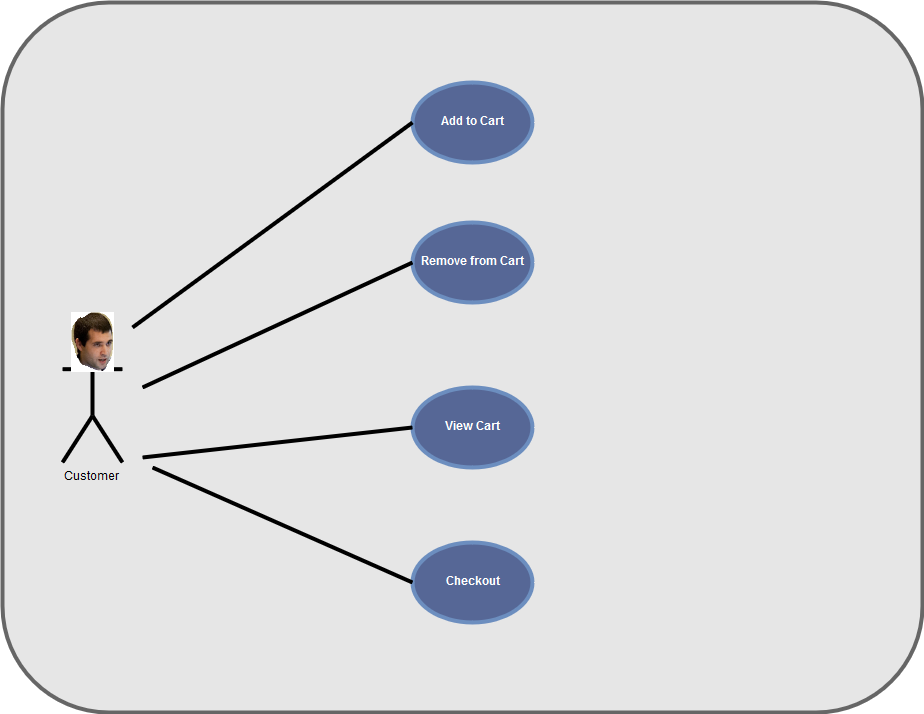
The Amazoom warehouse automatically manages a warehouse using autonomous robots, delivery trucks, and inventory trucks. The warehouse computer is the mastermind of the warehouse and takes orders from the web server, and if the desired item is in stock, will command robots to fulfill the orders by loading delivery trucks. As inventory runs low, the warehouse computer will automatically order inventory, and inventory robots will restock the warehouse as inventory trucks arrive. These robots will perform all duties by finding the shortest point, and by fulfilling duties from the delivery queue and the stocking queue. The warehouse manager can view details about the warehouse, through a UI that communicates with the warehouse computer. This includes checking order details, order status, and current inventory. As human beings are replaced by more skilled and dependable robot workers, Amazoom custom software rises to meet the demand. Why pay more for inconsistency, when perfection is at your fingertips.



**Use Cases**

This section covers the use cases for different users of the Amazoom warehouse and their user stories.

**Customer Use Case Diagram**



**Case 1: User Wants to Add an Item to Their Cart**

**Start of Case 1**

1. The user selects an item and enters the quantity
2. If the item is in stock, and the desired quantity is in stock, the items are added to the cart else **<<Scenario 1>>**
3. Item(s) are added to cart

**End of Case 1**

**Scenario 1: Return error with regards to item or quantity and display error to user. Nothing is added to cart.**

**Case 2: User Wants to Remove an Item to Their Cart**

**Start of Case 2**

1. The user can view the contents of their current cart
2. The user selects an item, and a specific quantity to remove. If the item is within their cart, and the quantity is valid, remove the item.
3. **If** quantity is invalid, then **<<Scenario 1>>**
4. Update Cart

**End of Case 2**

**Scenario 1: Prompt for a valid quantity**

**Case 3: User Want to View Cart**

**Start of Case 3**

1. The user selects to the option to view cart
2. The user’s cart is displayed listing all item names and quantities.
3. **If** empty **<<Scenario 1>>**

**End of Case 3**

**Scenario 1: “Cart is Empty” is printed on the screen**

**Case 4: User Wants to Checkout**

**Start of Case 4**

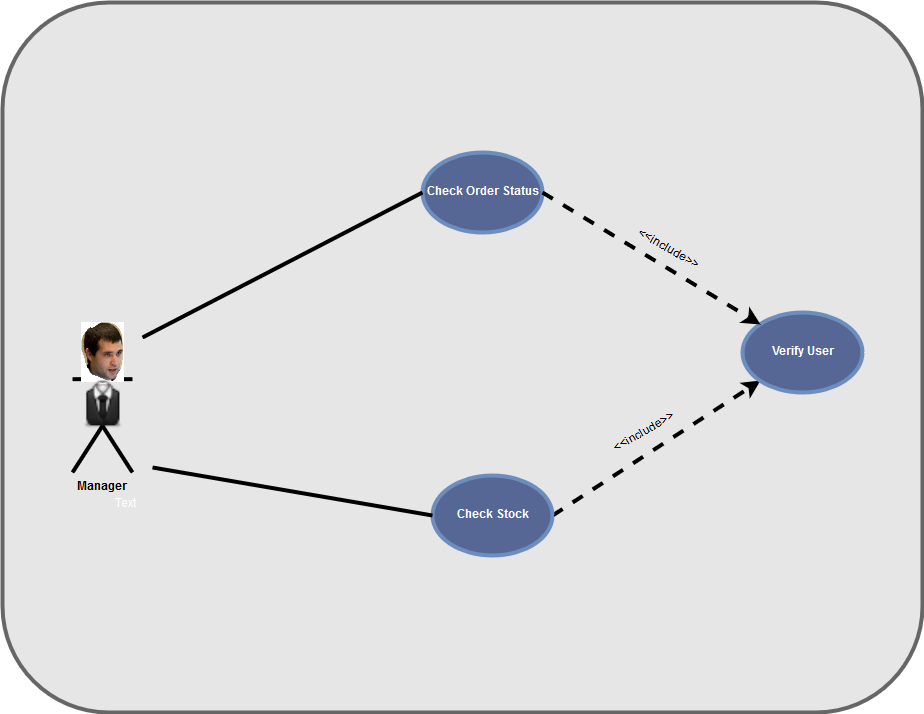
1. The user can checkout to purchase their items.
2. **If** cart is empty **<<Scenario 1>>**
3. The user checks out and order is sent from the web server
4. The order is then placed upon an order queue by the webserver, and order is verified
5. If order is verified continue, else **<<Scenario 2>>**
6. Order is successfully sent and the user is notified and receives an order number.

**End of Case 4**

**Scenario 1:** If the cart is empty display “Cart is Empty”

**Scenario 2:** Appropriate error is thrown i.e. “Insufficient Balance”. Transaction is cancelled.

**Manager Use Case Diagram**

****

**Case 5: Manager Wants to Query Status of an Order**

**Start of Case 5**

1. Manager logs onto computer using admin ID and password
2. Computer verifies admin ID and password
3. **If** valid continue, else **<<Scenario 1>>**
4. The user is prompted with options of
   1. Checking Status of Order
   2. Checking Stock of items
   3. Exit
5. User is prompted for order ID. User enters the ID
6. **If** Order ID is valid continue, else **<<Scenario 2>>**
7. The order is displayed listing the item name, quantity, and status
8. Computer exits to menu

**End of Case 5**

**Scenario 1:** Prompt for user name and password again

**Scenario 2:** Prompt for Order ID again

**Case 6: Manager Wants to Check Number in Stock of An Item**

**Start of Case 6**

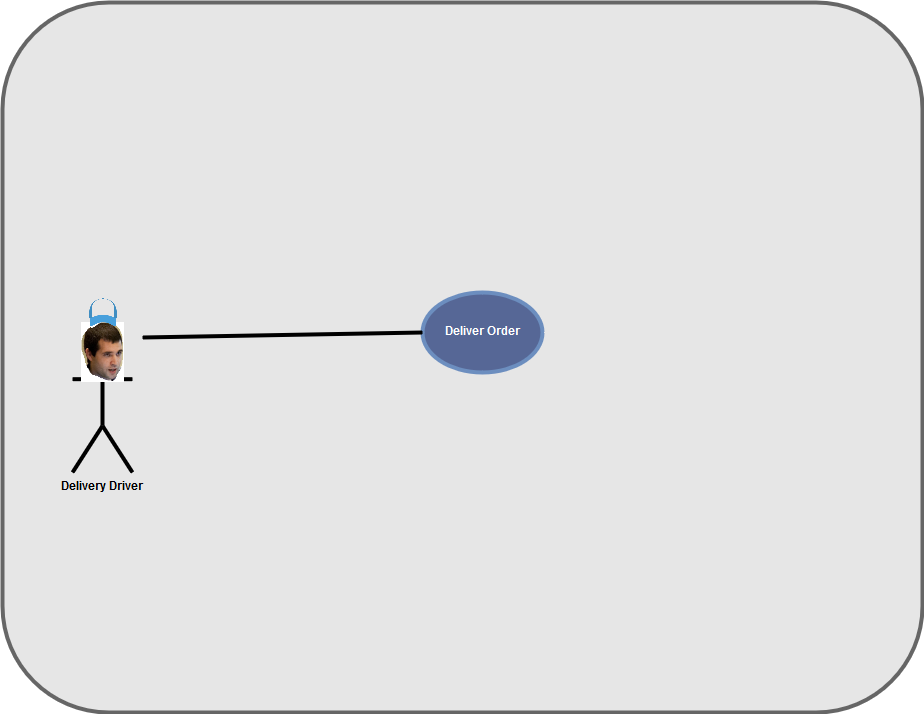
1. Manager logs onto computer using admin ID and password
2. Computer verifies admin ID and password
3. **If** valid continue, else **<<Scenario 1>>**
4. User selects check number of in stock items
5. User is prompted for a name, that is then used to search current stock of items.
6. The warehouse computer searches the stock shared memory for the ID.
7. If the item name is valid, return amount in stock else **<<Scenario 2>>**
8. Necessary information is displayed on the screen
9. Warehouse computer UI exits to main menu

**End of Case 6**

**Scenario 1:** Prompt for user name and password again

**Scenario 2:** Item doesn’t exist, return an error, and list all items and stock values, then exit

**Delivery Driver Use Case Diagram**

****

**Case 7: Delivery Driver Wants to Collect Deliveries**

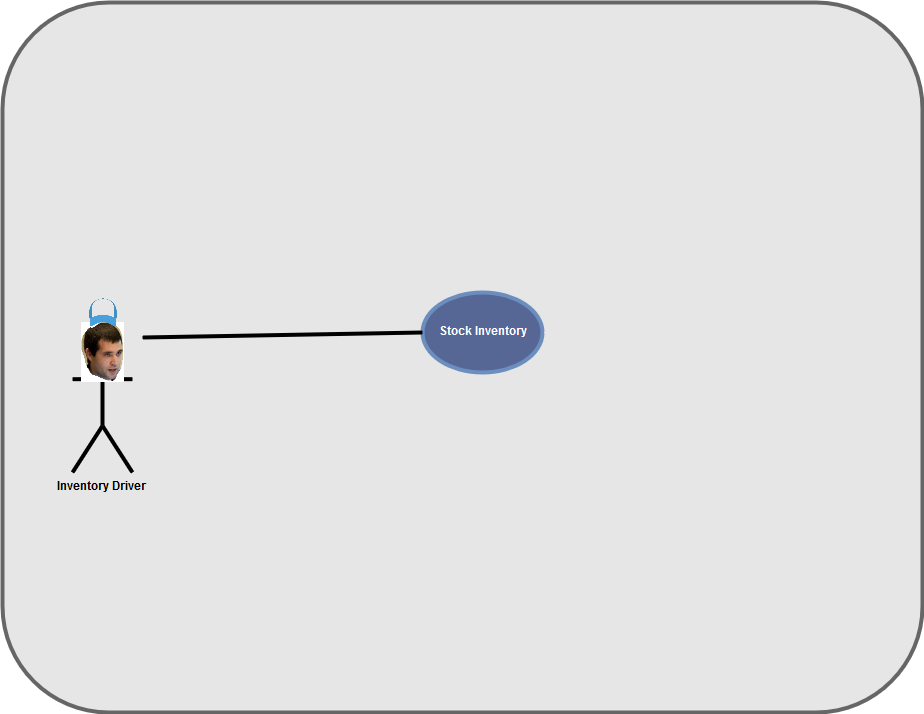
**Start of Case 7**

1. Wait on loading dock
2. If the dock is available continue, else **<<Scenario 1>>**
3. Once dock is available, park in loading dock and claim space
4. Wait until truck is full or no orders are left
5. Signal loading dock is free
6. Driver departs to deliver orders

**End of Case 7**

**Scenario 1:** Wait until available

**Inventory Driver Use Case Diagram**

****

**Case 8: Inventory Driver Want to Drop Goods Off at Factory**

**Start of Case 8**

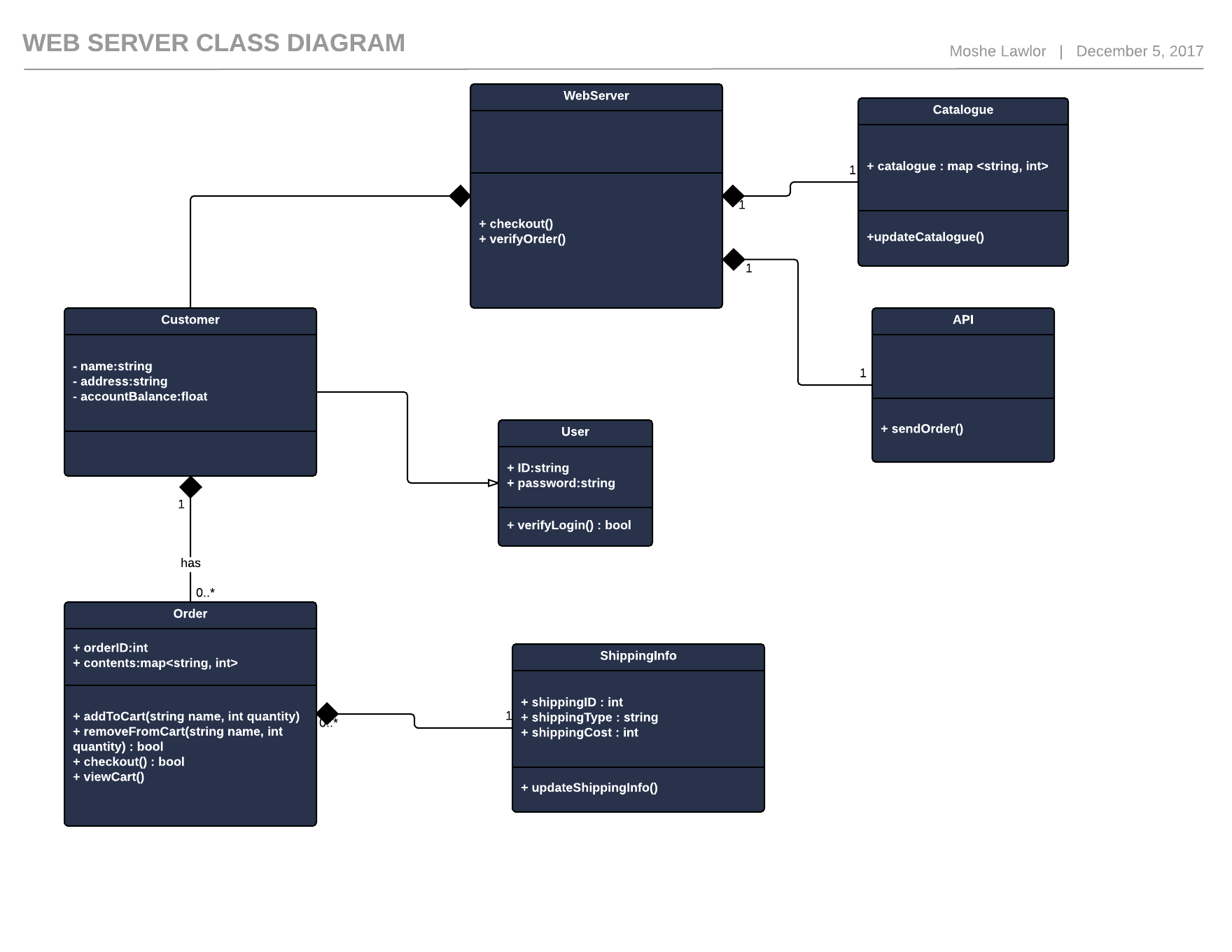
1. Wait on loading dock.
2. If the dock is available continue, else **<<Scenario 1>>**
3. Once dock is available, park in loading dock and claim space
4. Wait until truck is empty
5. Signal loading dock is free and leave

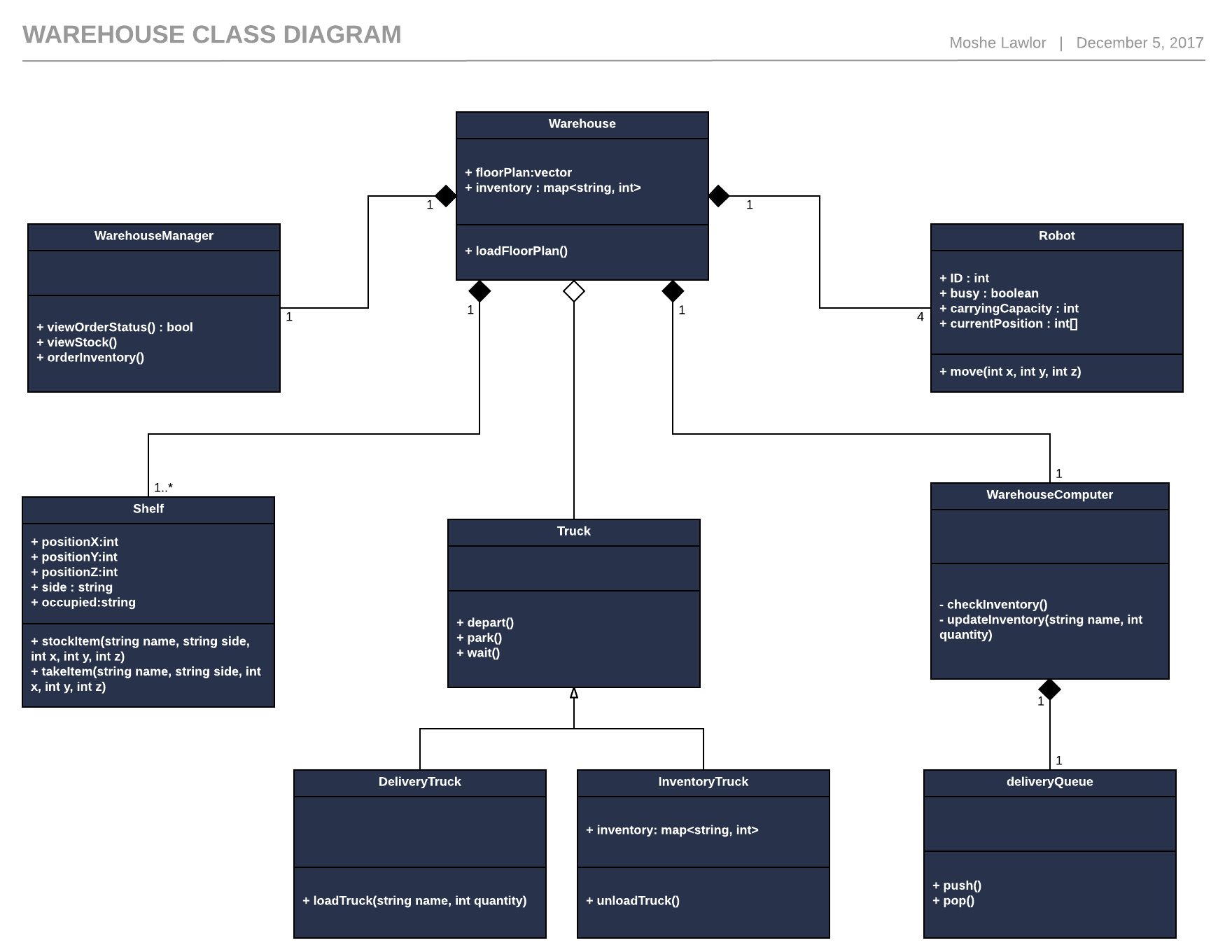
**End of Case 8**

**Scenario 1:** Wait until available

**Class Diagrams**

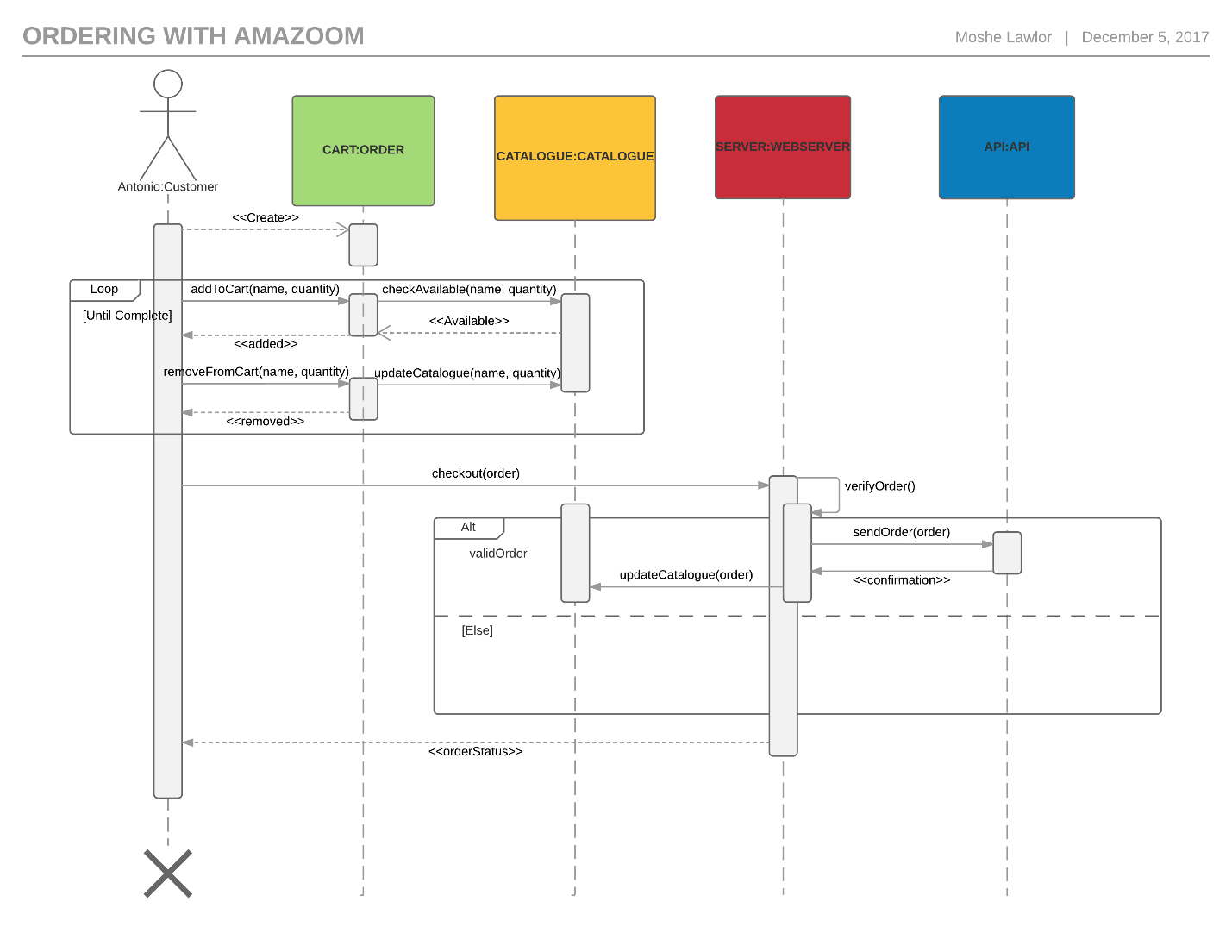
This section covers the class diagrams for the Amazoom warehouse. The class diagrams are split into two main diagrams, one for the warehouse, and one for the web server.

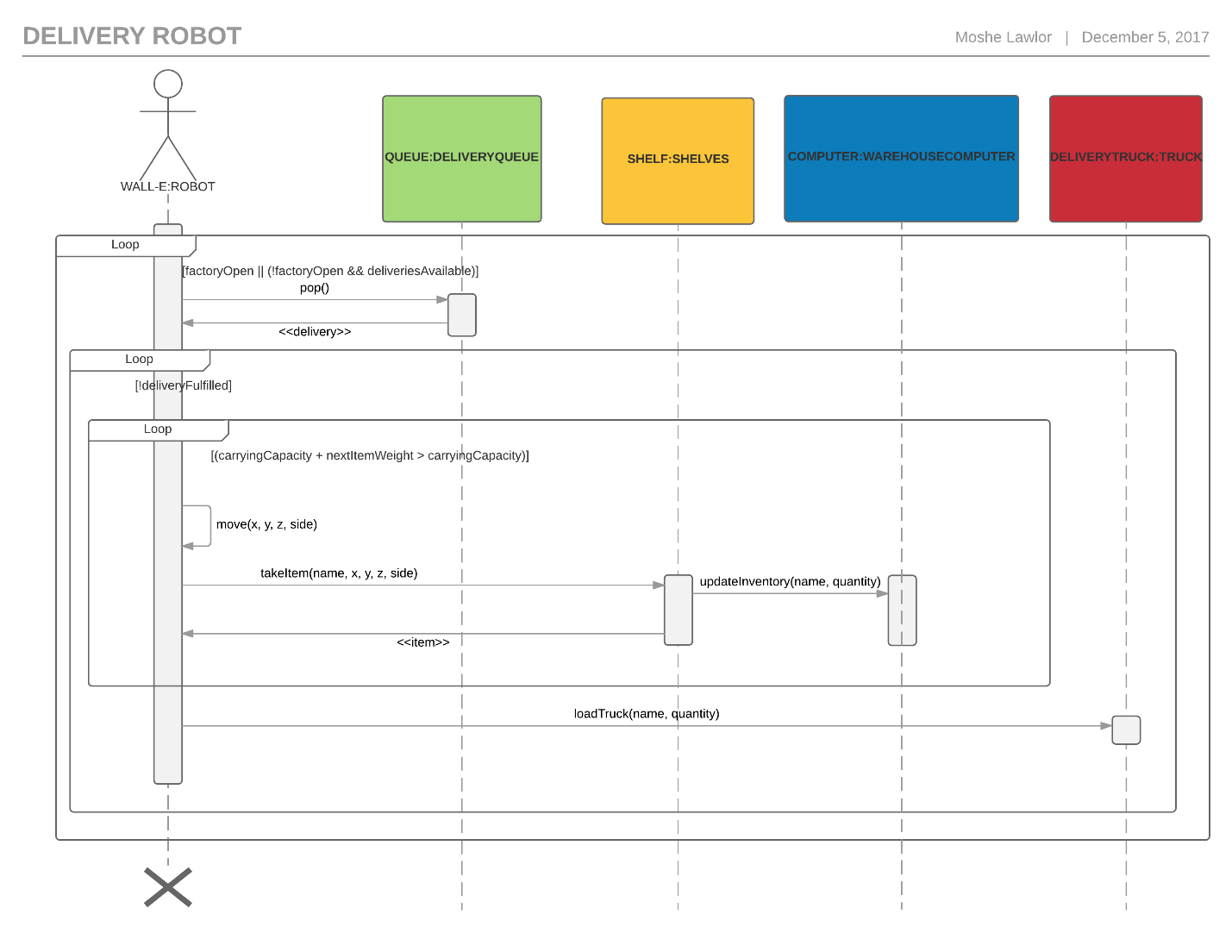


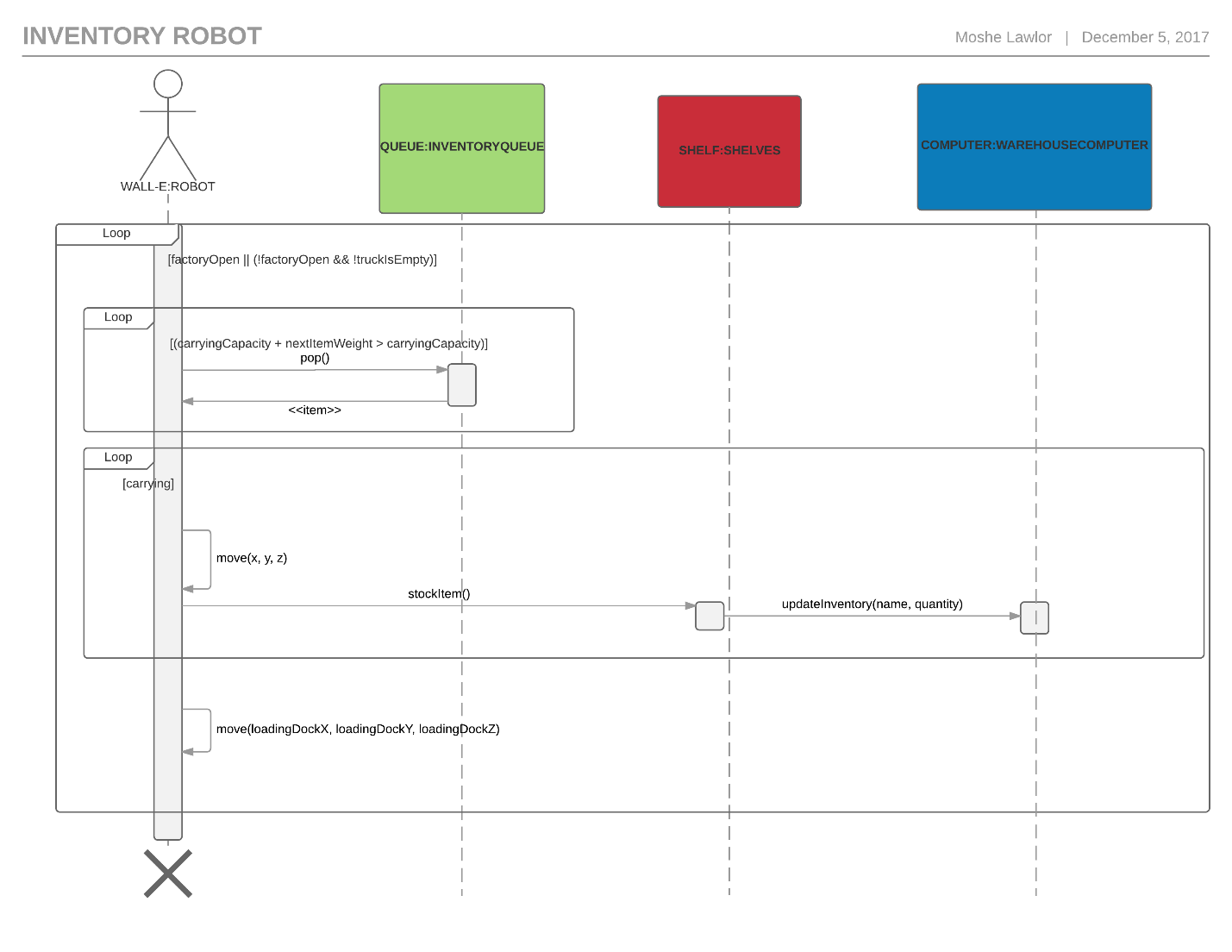


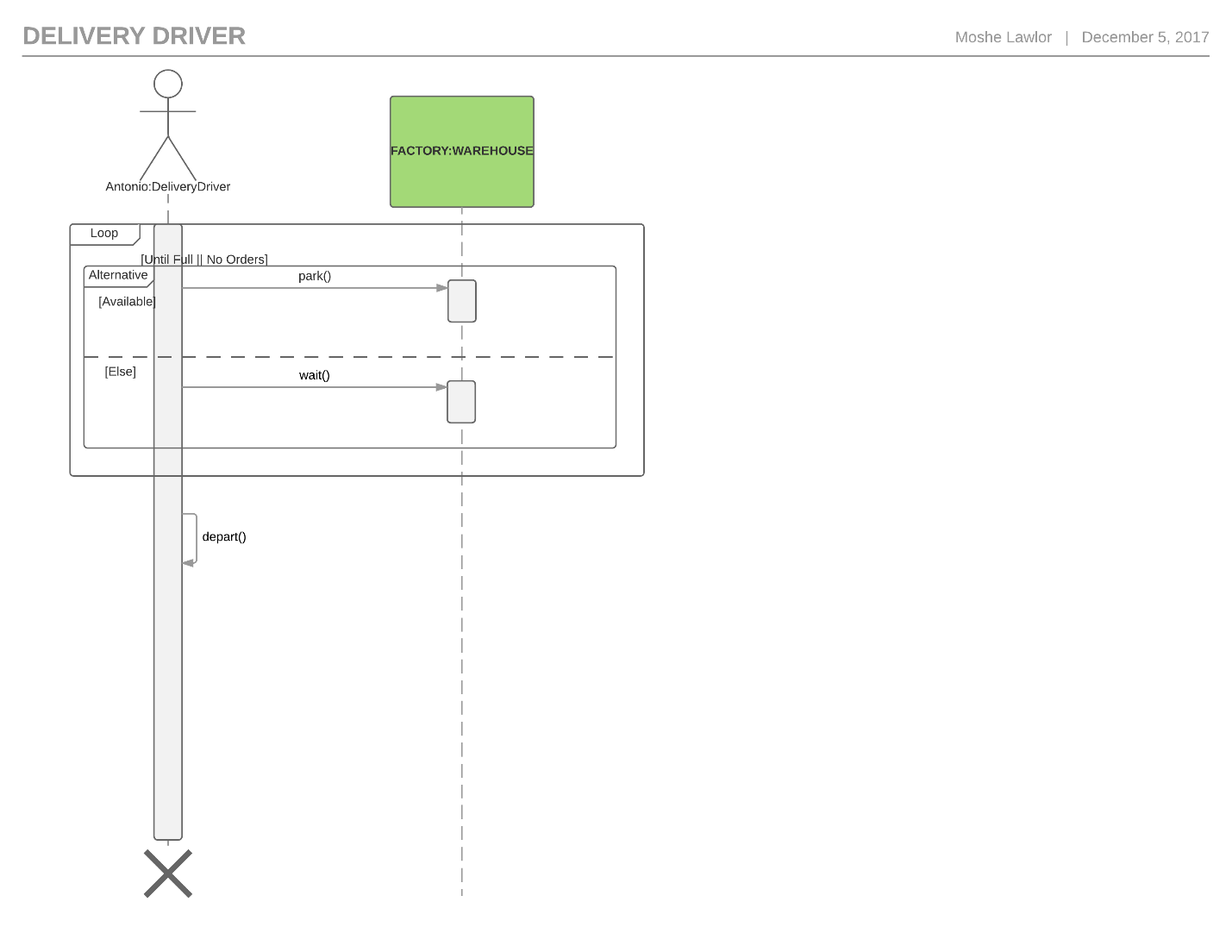
**Sequence Diagrams**

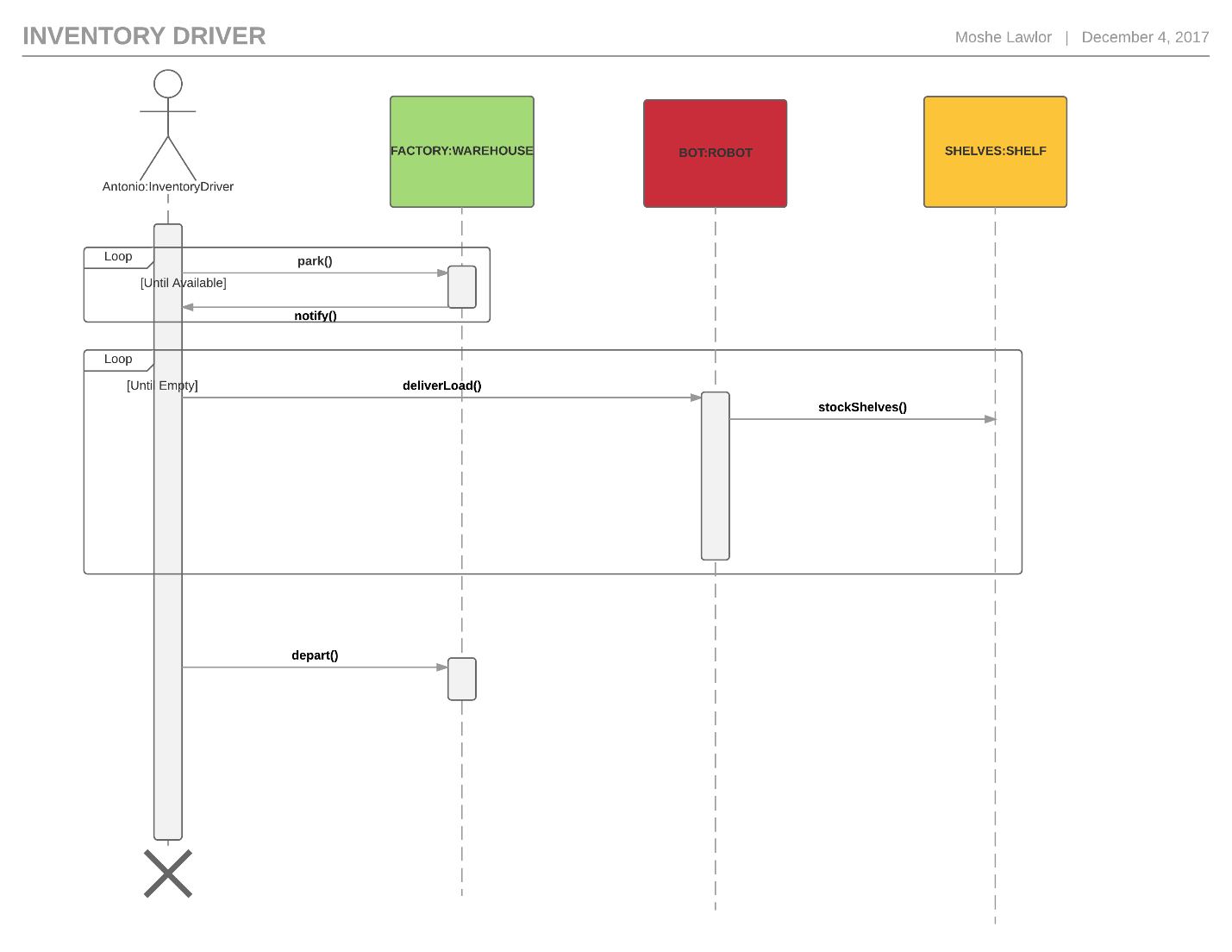
This section covers sequence diagrams for important interactions to illustrate exactly how the factory operates.





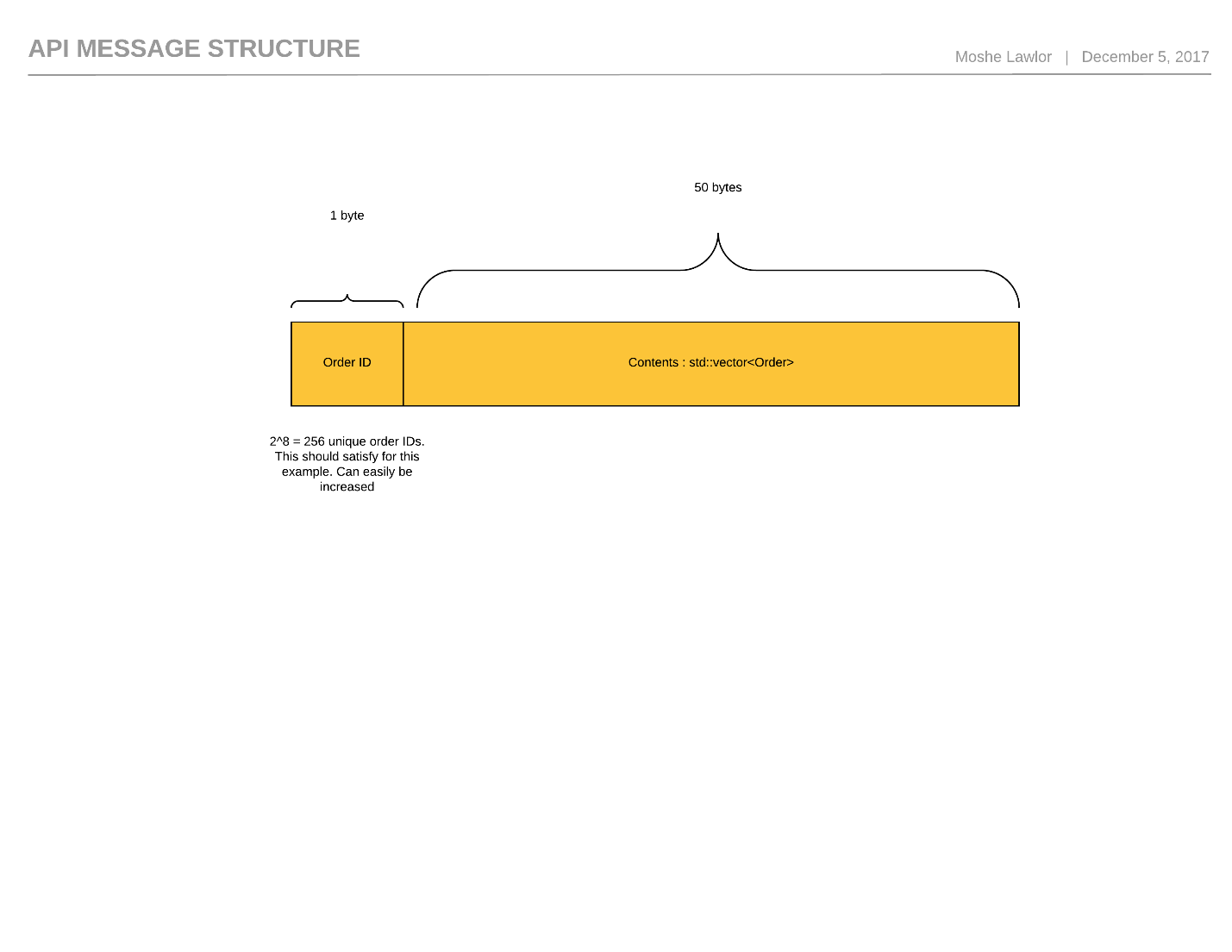






**API Communication Diagram**

This section shows the API communication diagram that was used to send orders to the server.



Note that there were some assumptions made about the maximum order size. To account for cases in which the order size is so large, we can simply split the order into multiple messages and have the warehouse computer update the order as it receives messages. In this case I chose a size that would allow a lot of items to be contained in the order. We simple convert the Order object to a JSON object to be able to send the message.