Release version v1.0
Oct. 2, 2023

CS 3311

Software Design

Fall 2023

Project Description

Kostas Kontogiannis

1. Introduction

The *Warehouse Management System* (WMS) is a software system that simulates the operations of the warehouse of a simple company that sells goods to clients and restocks products from suppliers.

1.1. Information Model

The model of the warehouse of the company can be implemented as a database with one or more tables denoting the ID of each product, the name of the product, its current stock quantity, its unit price, target max stock quantity for this product to be in the warehouse, target min stock quantity for this product to be in the warehouse, restock schedule (i.e. how many product units can be added per restock operation), and discount strategy ID.

1.2. Process Outline

Once a specific quantity of product is sold to a customer, this quantity is taken out of the available stock of this product from the corresponding database entry pertaining to this product. If the stock quantity of a product drops below the predefined min stock quantity set for a product, then the product's state is set as "*low stock*". If an order cannot be fulfilled because not enough products are in stock for this specific order, then the state of the product is "*restocking to fulfill order*"). Once a product's state is "*low stock*", or "*restocking to fulfill order*" then a notification is sent so a *restock* action can be triggered and the product is restocked. An order which leads for product to be on a "*restocking to fulfill order*" status (i.e. order quantity is higher than the available quantity), is fulfilled after the restocking operation for this product ends. Orders are sorted according to the time they were received. Earlier orders have to be fulfilled before orders received at a later time. Also, an order cannot exceed the target max stock quantity set for the product. If an order is placed which exceeds the target max stock quantity set for the product then the order is rejected by providing a message to the client.

Each product is restocked according to a specific restock schedule as specified in the database (see above). For example, a *restocking schedule* for a given product can be 50 units per restock operation, while for another product a different restocking schedule could be used, leading thus to a different restock quantities to be added for each product after each restocking operation. In addition to the restocking strategies, the price of an individual product for a given order can be calculated using different strategies implementing specific price discount business logic (e.g. orders of more than 100 units of this specific product get a 10% discount and if the

total value more than 1,000 dollars the order gets an additional 5% discount). Make sure you have more than 5 discount strategies.

The system will be based on a server process which will accept and process orders, and two or more processes which act as clients that place orders (see Fig. 1 schematic below). The communication between the client processes and the server process can be performed by any protocol of your choice. One suggestion is to use the Java http Server framework (see References below).

2. Use Cases

UC1: Login and Starting the Server

Once the server process starts by the administrator (i.e. invoking the main method e.g. in the Admin component – see Fig. 1) then the administrator has to provide a username and password combination (see Fig. 2). If the username and password combination is not correct, then the server cannot proceed and the process terminates. If the combination is correct the process continues by invoking the methods that start the Java http Server, displaying the server UI (see Fig. 3), connects to the product database, and allows for connections with client processes.

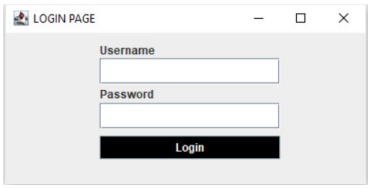

Fig. 2. Username and password UI

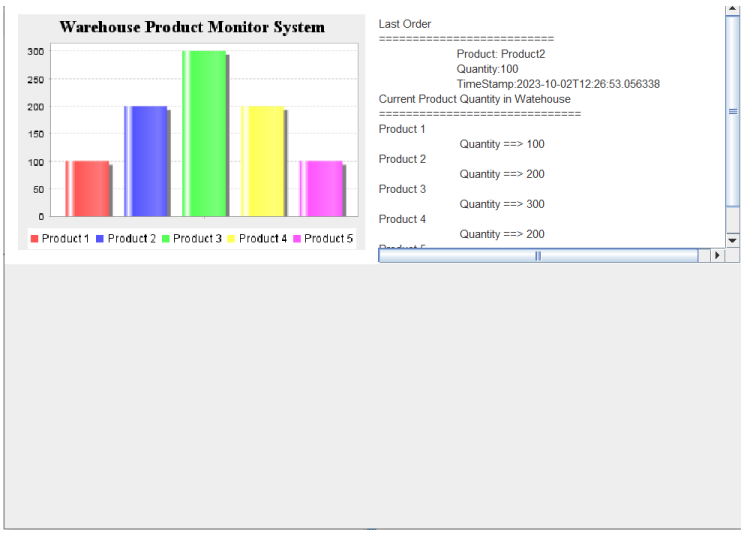

Fig. 3. Server UI depicting the current quantities in the Warehouse for five products.

UC2: Starting a Client and Connecting to Server

A client process can start directly without using a username and password combination by a user. Once the client process starts (see Fig. 4) it can connect to the server process to place orders and receive responses. A configuration file on the client provides the IP address of the server and the port number the server listens for incoming requests. The client-server communication can be achieved by using the Java http Server framework or any other framework of your choice.

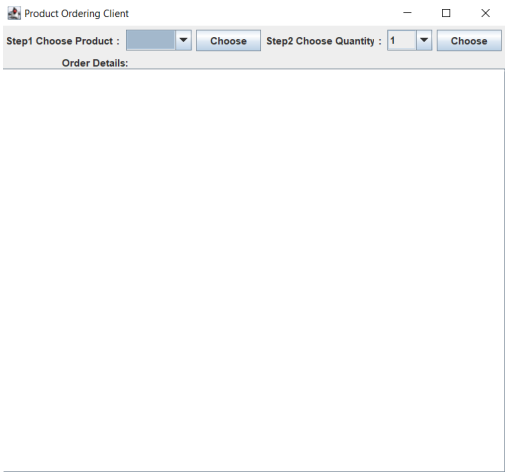

Fig. 4 Starting the client

UC3: Placing an Order

The client can select a product from a drop-down menu (see Fig. 4a) of available products and selects a quantity to be ordered.

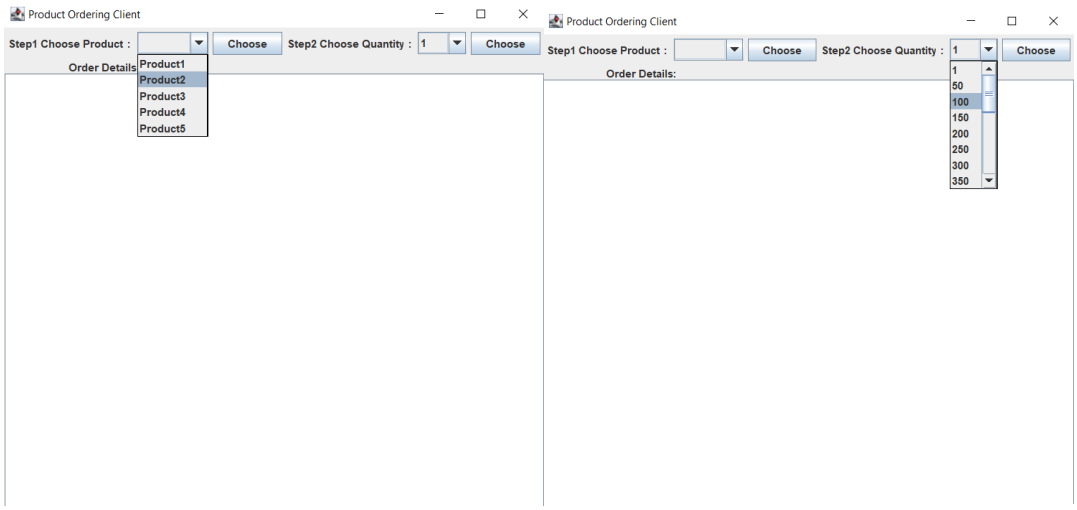

Fig. 4a Selecting first the product then quantity

When the order is placed, the client displays the message (see Fig. 4b) with the order details (i.e. product, quantity, and client timestamp).

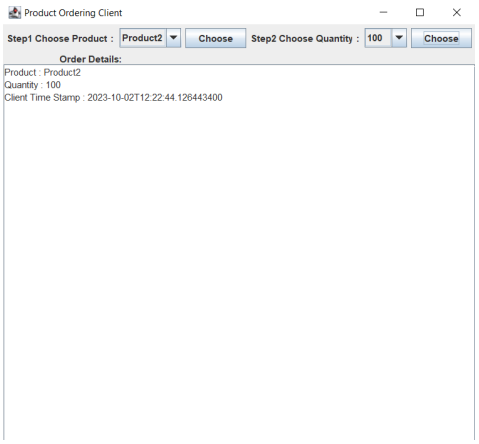

Fig. 4b. Client placing and order

UC4: Server Receiving and Processing Orders

When the order is placed, the server UI displays a message with the details of the last order received (i.e. this order – product, quantity, server's local timestamp) and the bar graph with current available quantities (see Fig. 3).

If the ordered quantity exceeds the target max stock quantity for this product in the warehouse, then the server will reject the order, and a response message from the server will be displayed in the client that the "*Order exceeds the max quantity set for this product and cannot be processed*". Also, the server UI displays the same message.

If the quantity exceeds the available quantity in the warehouse for this product at the time the order is processed by the server, a message is displayed in the server UI that the "*Order for Product X Quantity Y is pending – order exceeds available quantity".* In this case **Use Case UC6** is invoked. The client is nit notified at this point.

Once the order is fully processed then a response message from the sever is displayed in the client that the "*Order is finalized for Product X and Quantity Y with total price Z*" (replace X with product ID or product name, Y with order quantity, and Z with the total price computed by the appropriate pricing strategy). The product database is updated by reducing the quantity available by equal amount of the order fulfilled.

If after fulfilling and order, the quantity of the product in the warehouse drops below the predefined min stock quantity set for a product, then the product's state is set as "low stock" and a message is displayed in the server UI "*Restocking Operation for Product X initiated*" (replace X with the product ID or product name). In this case **Use Case UC6** is invoked.

Orders are processed in the order they are received. That is the orders are placed on a queue and are processed in sequence one after the other. In order to simulate the delay of fulfilling an order, you can add a random delay ranging from 30 to 45 seconds.

Example 1 of sequences of messages:

Client sends order with product quantity → client UI displays message "*Product X, Quantity Y, Timestamp Z (see Fig. 4.b* → (if order quantity is higher than the available quantity the server needs to restock) → server UI displays the message "*Order is received for Product X and Quantity Y*" (replace X with the Product ID or product name, and Y with the quantity ordered) → server UI displays the message "*Order for Product X Quantity Y is pending – order exceeds available quantity"* →server UI displays the message *Restocking Operation for Product X initiated*" → restocking finishes and server UI displays message "*Restocking Operation for Product X completed*" → server completes the order → client gets the message from server "*Order is finalized for Product X and Quantity Y with total price Z*".

Example 2 of sequences of messages sent to a client process:

Client sends order with product quantity → client UI displays message "*Order for Product X Quantity Y is sent*" → (if quantity is higher than max quantity set for this product in the warehouse) → UI displays the message "*Order is received for Product X and Quantity Y*" (replace X with the Product ID or product name, and Y with the quantity ordered) → server UI displays the "*Order for Product X Quantity Y exceeds the max quantity set for this product and cannot be processed*" → client gets the message from server "*Order for Product X Quantity Y exceeds the max quantity set for this product and cannot be processed*"

UC5: Product Pricing

The price for an order is based on a pricing strategy. An example of pricing strategy is as follows:

If an order quantity is of more than ***x*** number of units then a ***y%*** discount is applied, and If the total value of the order is more than 1,000 dollars, then an additional aggregate 5% discount is applied. This is specified by a strategy ID that refers to a specific business logic for determining the final price for an order. For example, the above strategy may be PricingStrategy001 (see factory method and strategy design patterns).

UC6: Restocking

The restocking process brings the product stock back to its max stock quantity as this is specified in the database. The restocking of a product is based on the *restocking schedule* of the product as also specified in the database. For example, if there must be 100 products to be reordered for the product o be restocked to its max quantity, and the restocking schedule is 30 products, them four restocking operations have to be performed, three operations of 30 products each, and one of 10 products. Once the restocking operation is completed then the server UI is updated with the new product quantities (see Fig. 3). Once a restocking operation starts the server UI displays "*Restocking Operation for Product X initiated*" and once it completes it displays "*Restocking Operation for Product X completed*" (replace X with the product ID or product name).

3. Overall System Structure

A schematic of the overall structure of the Warehouse Management System is depicted in Figure 1 below.

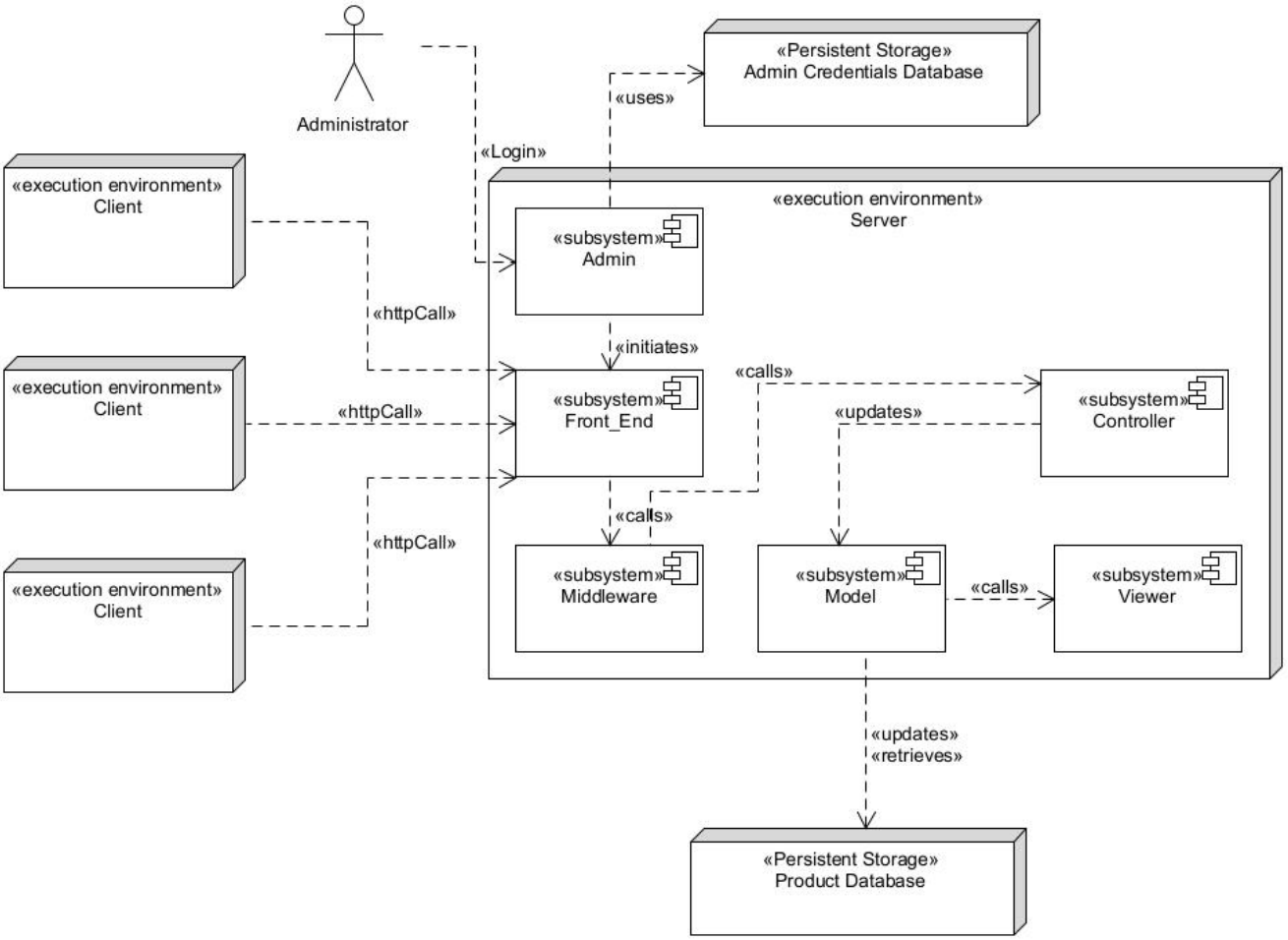

Fig. 1. System Structure Schematic

Client: A process that allows to send order requests to the server. It has a simple UI where:

a) the user can select the product to buy
b) the user can select the quantity of the product to buy
c) the server can send back responses

Server: A process that implements the back-end functionality of the Warehouse Management System. It can have a number of running components as follows:

Admin Subsystem: A component that allows the administrator to provide a User Name and Password combination and allow access to the functionality of the system. Once the Admin component starts it asks for credentials, and if the credentials are correct then the server starts. The Admin component can have other subcomponents (e.g. Login Server, Authentication Server).

Front_End Subsystem: It is a component that receives the incoming requests. It is where the handlers are defined.

Middleware Subsystem: A component that isolates the **Front_End** and the **Controller.** It oes some housekeeping such as keep a queue of incoming requests.

Controller Subsystem: A component that implements the business logic of the system (i.e. use cases UC4-UC6).

Model Subsystem: A representation of the run time data (e.g. Order, Product) – see the MVC pattern.

Viewer Subsystem: Displays the current warehouse product quantities (see Fig. 3).

Product Database: Keeps information about products on one or more tables (or files) denoting the ID of each product, the name of the product, its current stock quantity, its unit price, target max stock quantity for this product to be in the warehouse, target min stock quantity for this product to be in the warehouse, restock schedule (i.e. how many product units to be added per restock operation), and discount strategy ID.

Admin Credentials Database: Keeps information about valid username and passwords.

4. Sample Http Server and Client with URL parameters, and Client-side and Server-side UI

See example code in eclass.

NOTES

You should have up to 5 products in your product database, but no less than 2. You should have 2 or more pricing strategies, but no more than 5. A client can send multiple requests for different products one after the other (i.e. once one terminates). You can have multiple clients sending requests for the same product.

Applicable design patterns: Singleton, Proxy, Factory Method, State, Observer, Façade, Adapter

REFERENCES

Java http server:
https://dzone.com/articles/simple-http-server-in-java
https://www.rgagnon.com/javadetails/java-get-url-parameters-using-jdk-http-server.html
https://stackoverflow.com/questions/3732109/simple-http-server-in-java-using-only-java-se-api
http://www.microhowto.info/howto/serve_web_pages_using_an_embedded_http_server_in_java.html
https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/HttpServer.html