|  |
| --- |
| **AceMarket Business Enterprise Platform Project Report** |

**Florin Bordei, 280593**

**Kenneth Jensen, 280269**

**Dave Le, 280071**

**Jaume Lopez, 282231**

****

**Supervisor:**

**Steffen Vissing Andersen**

**VIA University College**

**24.679**

**ICT Engineering**

**2nd Semester**

**07/06/2019**

**Table of content**

[Abstract iv](#_Toc501021071)

[1 Introduction 1](#_Toc501021072)

[2 Requirements 2](#_Toc501021073)

[2.1 Functional Requirements 2](#_Toc501021074)

[2.2 Non-Functional Requirements 2](#_Toc501021075)

[3 Analysis 3](#_Toc501021076)

[4 Design 4](#_Toc501021077)

[5 Implementation 5](#_Toc501021078)

[6 Test 6](#_Toc501021079)

[6.1 Test Specifications 6](#_Toc501021080)

[7 Results and Discussion 7](#_Toc501021081)

[8 Conclusions 8](#_Toc501021082)

[9 Project future 9](#_Toc501021083)

[10 Sources of information 10](#_Toc501021084)

[11 Appendices 1](#_Toc501021085)

# Abstract

The aim and the objective of the project is to create a system, that enables the company AceMarket to work more efficiently and to reduce time-consuming tasks with the help of different functionalities implemented in this system. Therefore, the main objectives have been to facilitate the communication and transmission of information and items between the main actors inside of AceMarket. Later in the report will be described the requirements and the environment that AceMarket operates in.

In order to implement the requirements requested by AceMarket, the team has made use of different technical. System has been implemented in Java and supports three clients with one server. A usage of SQL database was required for persistent storage.

By using the above-mentioned technologies, the core functionalities have been implemented and the system complies with the requirements wished by AceMarket. The three main actors can communicate, make transfer of data and store the necessary information.

# Introduction

Operation efficiency and low-cost operations are fundamentals for the success of any company on the market that desires to be successful and expand to a larger audience of consumers and customers. Such characteristics would enable any company to succeed and stand out within its competitors. In our current days, companies are making more use of technology than before, as they realize, that with its help, they can achieve the above-mentioned attributes.

AceMarket is to be considered one of these types of companies, that need to improve their operations and business processes. AceMarket is a local retailer based in Horsens and it runs a family type business with the aim to provide local grown products to its consumers. The company has grown in popularity and it expects expansion in the near future. Despite this success, the company is facing internal challenges and a re-organization of the company’s working methods is needed before they decide upon future expansions. Currently, they are running the small business with a main office, called Headquarters; a warehouse, called Warehouse; and a shop, called Retailer.

In this setup they want to start implementing a system that will increase their efficiency and profitability. The Headquarters is the center of their operations and it lays down instructions for the Warehouse and Retailer on how to operate. Also, Headquarters is the main supplier of goods, for now, to the Warehouse. The Warehouse provides inventory and delivery of goods to the Retailer. That means, the Retailer is the entity, which completes the circle of goods through the sales operations. Retailer contains its own stock inventory, performs and stores sales operations data and it requests products from the Warehouse.

A key factor of AceMarket’s inability to perform is the lack of efficiency in relation to stock inventory and how this info in processed. Parts of the company are managing their stock inventory through multiple Excel spreadsheets when in other parts stock inventory management is absent. Therefore, it’s unable to synchronize the information across and keep an accurate stock inventory. This caused of the company financial loses internally and delays in deliveries that made some of their products unavailable for the market.

Another important issues that AceMarket is looking for a solution is related to their sales operation as they are using an outdated system that is unable to perform in a less time-consuming manner and it is unable to process any kind of information stored, such as insights into the purchases or consumer behavior. With this current situation the Retailer’s management is unable to deliver accurate performance reports to the Headquarters and in consequence, Headquarters are not able to perform any sales planning or predictions on long term.

Secondary problems are Human Resources functionalities as there is no working system at the moment that AceMarket can use to store their employee’s information that would enable the company to keep track of current active employees.

For these reasons, AceMarket has requested an interconnected system that would allow the company to have a general overview of their current affairs but also to be able to have data processing tools in relation to their inventory stocks and sales operations. The system must be able to facilitate stock item storage in all three entities, have possibility of updated overviews, request and delivery of goods and sales operations and analysis.

The system will have some delimitations as it will not have an invoicing system implemented and it will not be accessed by external entities. Also, the system will only be accessed by Java application on their computers. Another limitation would be delays in the data transmission between the entities. Last limitation to mention is the sales analysis implemented that is a rudimental one and it leaves space for future development.

Based on the company’s environment and requirements the next session of Requirements will lay out the implementation of functionalities and requirements of the system. The requirements have been extracted from the company’s requirements and the user stories gathered by the group’s team.

# Requirements

We have three main actors in this case, called as: Headquarters, Warehouse and Retailer. Each actor is performing similar functions but also, they distinguish in some case cases. For each actor the following user stories have been selected:

**Headquarters**

1. As a manager at the Headquarter, I would like to store my list of employees with info so I can have a clear overview over my human resources.
2. As a manager at the Headquarter, I want to be able to see the previous deliveries to the Warehouse so I can have an overview over this business process
3. As a manager at the Headquarter I would like to delete Stock Items from Headquarter inventory so that I can have an accurate inventory.
4. As a manager at the Headquarter, I would like to be able to store stock items so that I can have an accurate stock inventory.
5. As a manager at the Headquarter, I would like to be able to check my stock inventory so that I can have an overview.
6. As a manager at the Headquarter, I want to be able to edit my list of employees so that I can remove non-working employees.
7. As a manager at the Headquarter, I would like to be able to see all deliveries between Shop and Warehouse so I can have control of it and overview
8. As a manager at the Headquarter, I would like to see the Stock Inventory at the Shop so I can have an overview.
9. As a manager at the Headquarter, I would like to see the Stock Inventory at the Warehouse so I can have an overview.
10. As a manager at the Headquarter, I would like to see the list of employees from the Warehouse so I can have an overview.
11. As a manager at the Headquarter, I would like to see the list of employees from the Shop so I can have an overview.
12. As a manager at the Headquarter, I would like to be able to communicate with all managers so that we have an efficient communication.

**Warehouse**

1. As a manager at the Warehouse, I would like to be able to store stock items so that I can have an accurate stock inventory.
2. As a manager at the Warehouse, I would like to be able to check my stock inventory so that I can have an overview.
3. As a manager at the Warehouse, I would like to be able to store my employees so that I can have an overview of my active employees.
4. As a manager at the Warehouse I would like to be able to see my list of active employees so that I can have an overview.
5. As a manager at the Warehouse, I need to request products from Headquarters to keep my stock levels optimal.
6. As a manager at the Warehouse, I want to be able to edit my list of employees so that I can remove non-working employees.
7. As a manager at the Warehouse I would like to delete Stock Items from Warehouse inventory so that I can have an accurate inventory.
8. As a manager at the Warehouse, I want to be able to check the previous deliveries from Headquarter to Warehouse so I can have an overview.
9. As a manager at the Warehouse, I would like to check the previous deliveries from the Warehouse to the Shop so I can decide on future requests of products from HQ.

**Shop**

1. As a manager at the Shop, I need to request and receive products from Warehouse so I can maintain to avoid lack of inventory.
2. As a manager at the Shop, I would like to input and store all my sales units so I can have an overview of my Shop's sales efficiency
3. As a manager at the Shop, I would like to be able to see all my stored stock items so I can have an overview of my stock inventory levels.
4. As a manager at the Shop, I would like to be able to add items to Stock Inventory so I would be able to order them from Warehouse.
5. As a manager at Shop, I would like a sales analysis based on sales so I can have data to later report to Headquarter.
6. As a manager at the Shop, I need to have an overview of previous deliveries from the Warehouse so I can have an overview.

Based on the selected user stories, the following functional and non-functional requirements have been formulated:

## Functional Requirements

1. Headquarter manager must be able to store a list of employees.
2. Headquarter manager must be able to view previous deliveries to Warehouse.
3. Headquarter manager must be able to delete stock items from inventory.
4. Headquarter manager must be able to store stock items in inventory.
5. Headquarter manager must be able to see the Headquarter’s stock inventory.
6. Headquarter manager must be able to remove non-active employees.
7. Headquarter manager must be able to see all deliveries between Warehouse and Retailer.
8. Headquarter manager must be able to see stock inventory at the Retailer.
9. Headquarter manager must be able to see stock inventory at Warehouse.
10. Headquarter manager must be able to see employee’s list at Warehouse.
11. Headquarter manager must be able to see employee’s list at Retailer.
12. Headquarter manager must be able to communicate with Warehouse manager and Retailer manager.
13. Warehouse manager must be able to view stock inventory at Warehouse.
14. Warehouse manager must be able to store stock items in inventory.
15. Warehouse manager must be able to store employee list at Warehouse.
16. Warehouse manager must able to view employee list at Warehouse.
17. Warehouse manager must be able to remove non-active employees.
18. Warehouse manager must be able to remove stock items from inventory.
19. Warehouse manager must be able to request and receive stock items from Headquarter.
20. Warehouse manager must be able to view previous deliveries from Headquarter to Warehouse.
21. Warehouse manager must be able to view previous deliveries from Warehouse to Shop.
22. Retailer manager must be able to request and receive products from Warehouse.
23. Retailer manager must be able to input and store sales.
24. Retailer manager must be able to store stock item at Retailer inventory.
25. Retailer manager must be able to view stock items in inventory.
26. Retailer manager must be able to access sales analysis.
27. Retailer manager must be able to view previous deliveries from Warehouse to Retailer.
28. The system must be able to extrapolate the current date and time.

## Non-Functional Requirements

1. The system must be developed in Java
2. The system must have an integrated database
3. The system should make use of the database
4. The system must contain Client/Server infrastructure

# Analysis

In this section are in-depth descriptions of three main entities of our project; Adding and storing stock items in the headquarters, requesting and receiving stock items at the warehouse, and lastly sales alongside with analysis at the shop. To visually show the flow of activities in the system, the activity diagrams were used, all of which have been derived from the relevant use case descriptions. The descriptions can be found in the appendix.

A close up of a sign

Description automatically generated One of the most important functionalities in the system is adding and storing stock item, which is essential to get data into the system. Adding the items requires the user to input the relevant information, like the name of the item, the cost, the quantity, the minimum and maximum stock the user wants to store. In headquarters department, the item gets an ID automatically, allowing it to be uniquely identifiable. All of this is shown in the activity diagram shown to the left.

Once the item is added, it gets stored in the database automatically, ensuring that no data is lost for any reason. The storage is achieved by connecting the user to a server, which stores the item in the database.

Adding and storing stock items is slightly different for the other users. If the managers want to add and store a stock item, the item needs to exist in the headquarters. They need to manually input ID, which needs to correspond with the generated ID from the headquarters, so that the item can still be uniquely identifiable when accessed over multiple clients. This will need to be done either through direct communication or through the implemented message A close up of a piece of paper

Description automatically generatedfunctionality.

Next, the system consists of requesting and receiving items from entity to entity. The explanation relates to warehouse manager case.

The activity diagram, that can be seen to the left, describes how to request a list of items. First, the client must add items to the list (see the appendix). Once the items are added, client has a possibility to change the list. Changing the list involves, for example, changing the amount to request or completely removing an item.

Once the client is satisfied with the list of items, he can send the request by pressing confirm and the list is sent to the server.

Then, the server changes the values in the database by increasing the requested item’s quantity by the requested amount and sends a message to the headquarters, that he should decrease all requested items’ quantities.

The sales process is a unique functionality accessed only by retailer manager. This part is split in 2, inputting and storing the sales and the sales analysis.

A screenshot of a cell phone

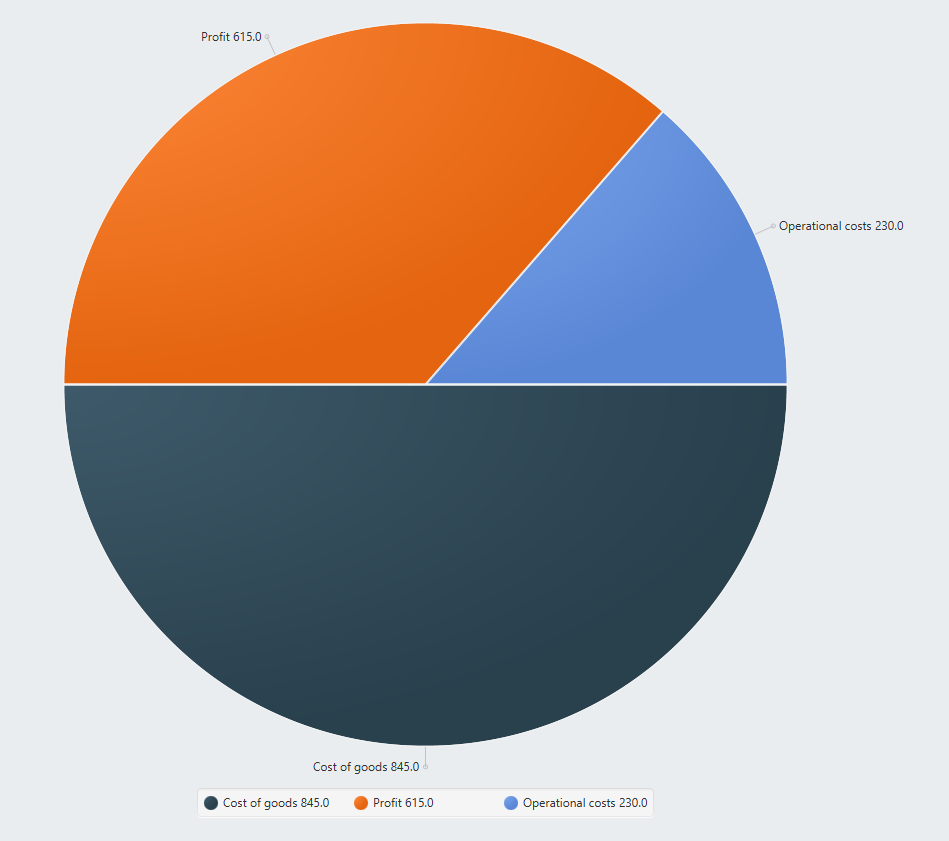
Description automatically generatedTo the left is shown the activity diagram of inputting sales. When the user sells an item, the system stores the change and sends it to the server. Then, the server stores it in the database, so that the client has a log of all sales.

The user will be able to see the new values, and the system will analyze the sales so far. This will be described on the next page.

A screenshot of a video game

Description automatically generatedThe activity diagram of this looks deceptively simply, because most of the work happens behind the scenes.

When the user presses on dashboard, the system shows the calculated analysis in the form of a pie chart. This chart shows shop’s current performance. A sample pie chart can be seen below.



As can be seen from the pie chart, it shows the ratios of profit, operational costs and the cost of goods, allowing the client to reflect on the performance of their shop and make decisions to increase their profits. Cost of goods represents the cost of the materials used in creating the good along with the direct labor costs used to produce the good. The retailer always sells items for double the cost of goods, during which time each item needs operational costs – costs to maintain the existence, which is equal to 1 dkk for each item. The profit can be got by deducting cost of goods and operational costs from the selling price.

These key features, as well as the rest of our requirements have led to this domain model:

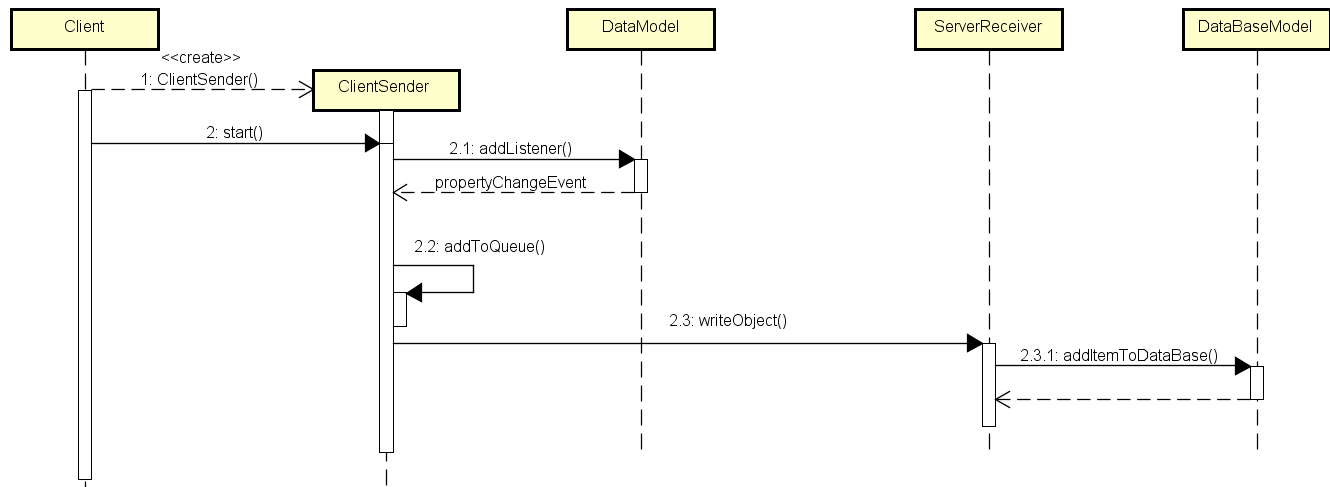
A screenshot of a map

Description automatically generated

Stock items stored in a list as well as a product request list. These are then sent via a delivery list to the relevant entity. Warehouse can send request to headquarter and retailer can send request to warehouse.

# Design

In this section, the same three main features as the analysis will be described, only going into more detail and using sequence and class diagrams.

First, we have the Headquarters adding and storing stock items. The flow through the system is described in the sequence diagram shown to the left.

It is shown on this diagram, that when the user inputs the data, the stock item must be stored. This is achieved by storing the values in the view in fields, passing said fields to the view model, which will store the data in the data model. This means that the system will implement the MVVM design pattern to achieve this flow of data. The way we then update the view is by use of listeners on the data model, causing the main inventory view model to update its fields, which are bound to the view. The way we send the data is by adding the data to the client sender queue when the event fires. The specifics of data transfer will be discussed later in this section.

A screenshot of a map

Description automatically generatedThe next sequence diagram that will be explored is the diagram for requesting and receiving items as the Warehouse manager. Said sequence is shown on the left.

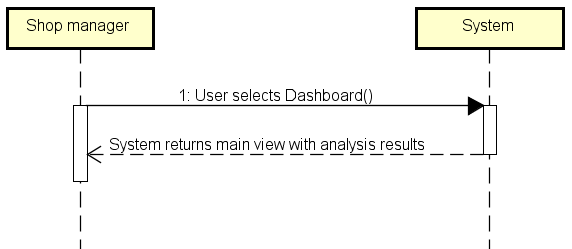
This sequence shows that as the user opens the list, they have two choices. If they want to edit, they can either remove or edit the requests in the list. All of this should be handled locally through the MVVM controls, but when the user presses confirm, the list should be packed up and sent to server.

A screenshot of a cell phone

Description automatically generatedThe last feature to be detailed is the selling of goods in the Retailer and the following analysis. The first sequence is for inputting and storing sales. This sequence is shown to the left.

What should happen when the user selects sell is, that the selected stock item, along with the quantity should be stored in a new Object. The quantity should then be removed from the stored stock item, to indicate that some amount of said item has been sold.

The new Object should then be sent to the server and the sale should be recorded in the database, along with the change in quantity of the original stock item.

With this stored, we can now move on to the sales analysis. The sequence diagram is quite simple as most of the work is done behind the scenes.

The work that should be done is, that for each item sold, the system should calculate the price of buying the item, the profit for selling the item and the operational costs. The operational costs have been set at 1 krone per item, as more precise values would require additional input from the stakeholders. These values should then be presented in a pie chart on the dashboard, with proper labels. The values should be displayed in the area of the graph, so if the profit fills the top-left part of the pie chart, the value of the profits should be shown there as well.

Now the client-server infrastructure will be explained. Both the client and server should create two classes, a sender and a receiver. The senders should contain a queue, acting as the consumer in a producer/consumer pattern. The producer would be the respective model, data model for client and database model for server. The sender and receiver should implement the Runnable interface to allow for multiple clients to connect to the database at one time.

The server receiver should implement a reader/writer class that handles the communication with the database to ensure that no data is being read while something is being written to the database. This should be safe, with priority for the writers, as changing the data as soon as possible is more important than reading from the database. The reader/writer class should be shared across all instances of the server receivers.

The communication between the server and client should be done by sending a packet object, containing a String representing the action desired from the receiver and a JSON string with the needed information to create the class needed for the operation.

All of this is shown in the partial class diagram shown on the next page.

A close up of a piece of paper

Description automatically generated

The arrows going out to the left is connecting to the data model, showing the association between the client sender and receiver and the data model. The same is true for the arrow going out of the bottom, except it is connected to the read/write class.

A screenshot of a cell phone

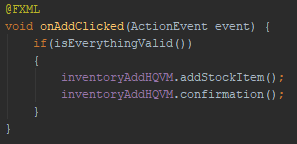
Description automatically generatedAs the whole class diagram is too big to show in this format, for more details please see the appended astah file containing the complete diagram in the “Analysis and Design” folder.

The database has been set up following the E/R diagram shown on the left.

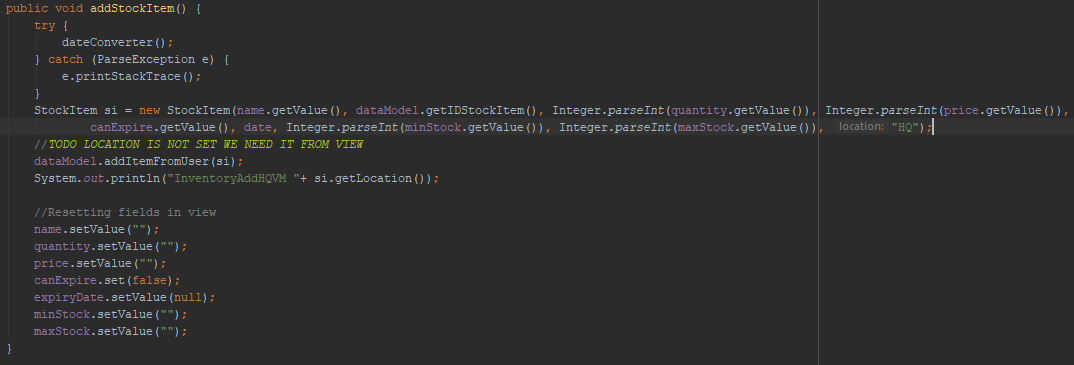
# Implementation

In this section, a complete path from adding a stock item to it being stored in the database will be shown in code. This will be from the perspective of the Headquarters, but the process is the same for all clients.

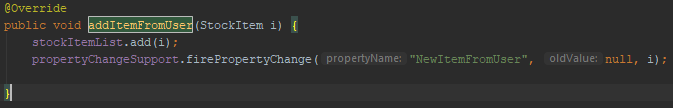
We start in the view class, when the button is pushed with the data inserted. The button press will call this method



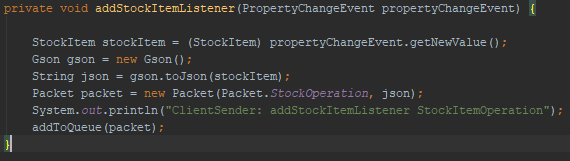
We will continue with the first method call, as all the second method does is create an Alert, telling the user that the item has been added. The addStockItem method looks like this



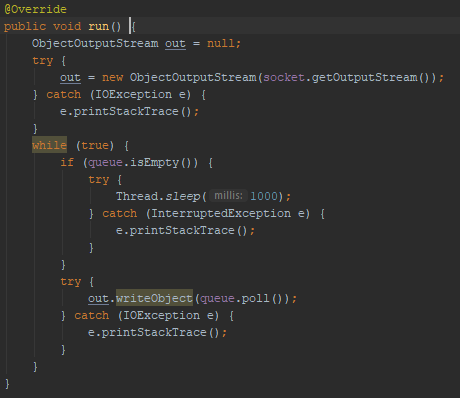
First it creates the stock item from the stored fields, then calls to the data model. Finally, it resets the fields in the view to allow the user to input another item seamlessly. Going into the data model, the method looks like this



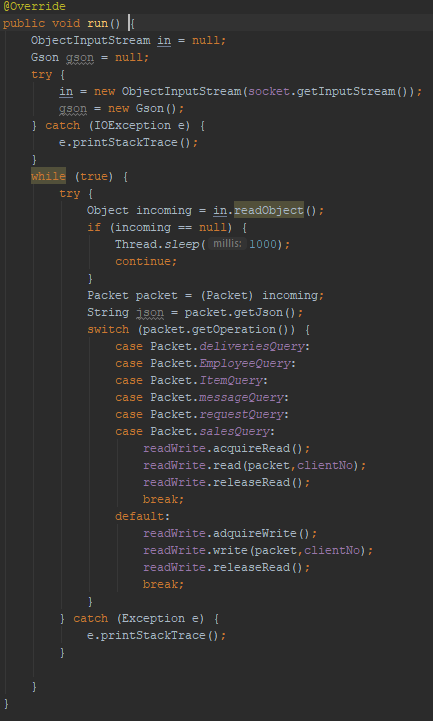
It adds the given stock item to the stored list, then first a property change event. This event has a listener in the client sender, meaning it will call this method



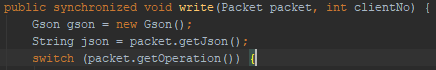
This will create a packet object and add it to the queue. This will be sent in this method



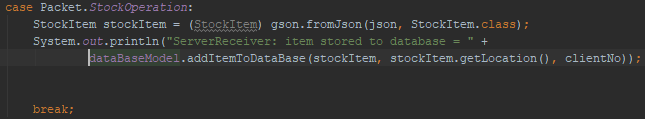
The output stream created using the socket will allow the sender to send the packet a server receiver. When the server receives it, this will happen



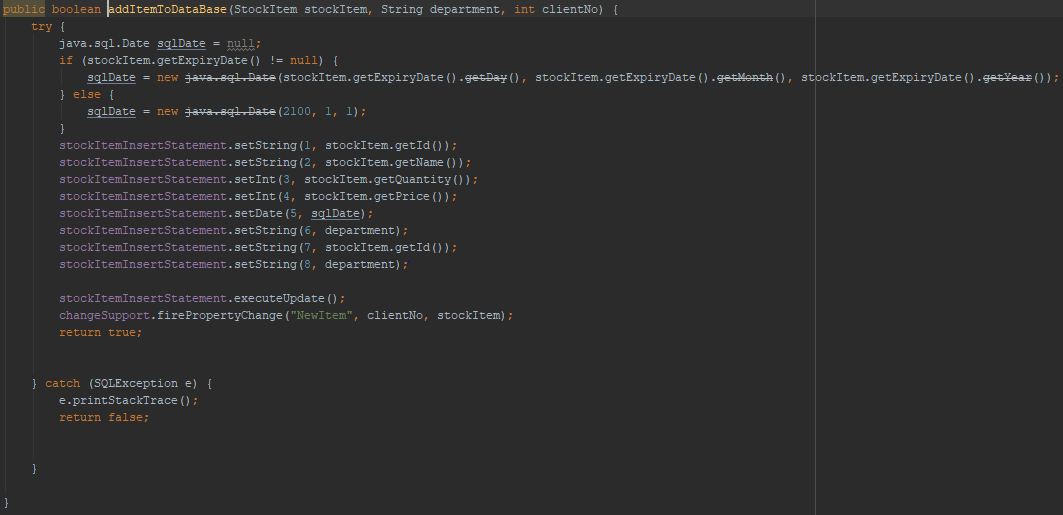
When the packet object was created, it was passed the Packet.stockOperation, so going into the default case, we can see that it acquires write access. This is done using the readers/writers pattern. When the writing is done, the thread will then release it’s access. The write method, however looks like this



As the shown switch is very large, only the relevant case will be examined



We make sure to print the results to the console to allow for debugging, but going into the database model, we can see that the method called is



This is where the prepared statement is created and is sent to the database. Now the system will exit back out through our methods, releasing the access on write and notifying other clients to update their lists, if necessary.

# Test

The tests performed on the system were Junit testing on the model and scenario testing for wider functionality. The system has passed all the tests given to it.

## Test Specifications

The tests have been performed in Java 8 using Junit 4.

# Results and Discussion

The project is a robust stock inventory system, with the possibility of communication between entities allowing for greater operational efficiency and profitability for AceMarket. The system is fully functional as-is, but can still be developed further in the future, should the need ever arise.

The use of various design patterns through analysis and design has led to the core functionality of the system being easy to serve the needs of AceMarket as well as easing the future development processes.

As mentioned earlier, with the help of analysis tools it had been understood, that the requirements around objects, including both behaviours and data created, had been modelled after the real-world objects, that the system interacts with.

In the design, the team has kept in consideration the limitations and needs of the customer and designed the system accordingly to make sure, that the finished project will align with the customers wishes.

Specifically in the data transfer, the use of producer / consumer pattern allowed for continuous checking of whether the user wants to send something, making the process of sending your data to the server seamless.

# Conclusions

The team has set up requirements with the stakeholders and has throughout the development process used their knowledge of software engineering methods to fulfill as many requirements as possible.

The completed system fulfills the most important of requirements and allows for the rest to be completed in future development.

# Project future

As a future for the project, it could available from the outside, either outside the system or from a non-java application. This would allow more flexibility for the user in accessing the system.

The database model could have implemented a state pattern, to allow for more efficient database access, and a visitor pattern could have been implemented in the model, to allow changes to be made without altering the existing model. The visitor would have allowed also extra features to be implemented.

The scalability of the system could be improved by improving the server-client infrastructure allowing for more clients of any type and for more servers to be added in tandem. This would facilitate the expansion of the customer’s business into multiple warehouses and retailers.

# Sources of information

Banger, D., 2014. A Basic Non-Functional Requirements Checklist « Thoughts from the Systems front line.... Available at: https://dalbanger.wordpress.com/2014/01/08/a-basic-non-functional-requirements-checklist/ [Accessed January 31, 2017].

Business Analyst Learnings, 2013. MoSCoW : Requirements Prioritization Technique — Business Analyst Learnings. , pp.1–5. Available at: https://businessanalystlearnings.com/ba-techniques/2013/3/5/moscow-technique-requirements-prioritization [Accessed January 31, 2017].

Dawson, C.W., 2009. *Projects in Computing and Information Systems*, Available at: http://www.sentimentaltoday.net/National\_Academy\_Press/0321263553.Addison.Wesley.Publishing.Company.Projects.in.Computing.and.Information.Systems.A.Students.Guide.Jun.2005.pdf.

Gamma, E. et al., 2002. *Design Patterns – Elements of Reusable Object-Oriented Software*, Available at: http://books.google.com/books?id=JPOaP7cyk6wC&pg=PA78&dq=intitle:Design+Patterns+Elements+of+Reusable+Object+Oriented+Software&hl=&cd=3&source=gbs\_api%5Cnpapers2://publication/uuid/944613AA-7124-44A4-B86F-C7B2123344F3.

IEEE Computer Society, 2008. *IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation*,

Larman, C., 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*,

Mendeley.com, 2016. Homepage | Mendeley. Available at: https://www.mendeley.com/ [Accessed February 2, 2017].

YourCoach, S.M.A.R.T. goal setting | SMART | Coaching tools | YourCoach Gent. Available at: http://www.yourcoach.be/en/coaching-tools/smart-goal-setting.php [Accessed August 19, 2017].

MSP Guide. 2012. Tuckman model. [ONLINE] Available at: <http://www.mspguide.org/tool/tuckman-forming-norming-storming-performing>. [Accessed 2 June 2019].

E-Stimate. 2015. E-Stimate model. [ONLINE] Available at: <https://www.e-stimate.com/wp-content/uploads/sites/16/Doris-Muster-.pdf>. [Accessed 3 June 2019].

Scrum Guide. 2018. Scrum Guide. [ONLINE] Available at: <https://www.scrumguides.org/scrum-guide.html>. [Accessed 4 June 2019].

Conolly, T. and Begg, C., 2010. *Database Systems*. 5th ed. US: Pearson Education.

Larman, C., 2004. *Applying UML And Patterns*. 3rd ed. US: Pearson Education.

jrebel.com. 2014. Object-oriented design principles and the 5 ways of creating SOLID applications. [ONLINE] Available at: <https://jrebel.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/>. [Accessed 15 May 2019].

docs.oracle. 2017. Lesson: All About Sockets. [ONLINE] Available at: <https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>. [Accessed 15 May 2019].

https://www.ibm.com. 2006. Traceability from Use Cases to Test Cases. [ONLINE] Available at: <https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html>. [Accessed 20 May 2019].

docs.oracle. 2019. Guarded Blocks. [ONLINE] Available at: <https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html>. [Accessed 13 May 2019].

sourcemaking.com. 2019. Observer Design Pattern. [ONLINE] Available at: <https://sourcemaking.com/design_patterns/observer%E2%80%8B%E2%80%8B>. [Accessed 13 May 2019].

via.itslearning.com. 2019. The Observer Design Pattern (SVA, February 2019).pdf. [ONLINE] Available at: <https://via.itslearning.com/LearningToolElement/ViewLearningToolElement.aspx?LearningToolElementId=364658>. [Accessed 13 May 2019].

en.wikipedia.org. 2019. State pattern. [ONLINE] Available at: <https://en.wikipedia.org/wiki/State_pattern>. [Accessed 13 May 2019].

sourcemaking.com. 2019. Visitor Design Pattern. [ONLINE] Available at: <https://sourcemaking.com/design_patterns/visitor>. [Accessed 13 May 2019].

geeksforgeeks.org. 2019. Java Singleton Design Pattern Practices with Examples. [ONLINE] Available at: <https://www.geeksforgeeks.org/java-singleton-design-pattern-practices-examples/>. [Accessed 13 May 2019].

# Appendices

All the required information and documentation can be found in the following folders:

* “Documentation” folder
* “Analysis and Design” folder
* The code can be found in the “SEP2” folder