**Daniel Imparatelu**

**1509309**

**Prediction of Sales and Management of Stock using Linear Regression and a Mobile Barcode Scanner – a Java-Based Inventory System**

BSc (Hons) Computer Science and Software Engineering

Undergraduate Thesis Report

Department of Computer Science and Technology

University of Bedfordshire

**Supervisor: Dr Mitul Shukla**

AY 17/18

Abstract

Acknowledgements

Key Words

**Number the pictures**

Contents

[1. Introduction 5](#_Toc511752214)

[**1.1.** **Aims and Objectives** 5](#_Toc511752215)

[**1.2.** **Techniques for realisation** 6](#_Toc511752216)

[**1.3.** **Thesis Structure** 6](#_Toc511752217)

[2. Literature Review 7](#_Toc511752218)

[**2.1.** **Introduction to the Literature Review** 7](#_Toc511752219)

[**2.2.** **Relevant studies** 7](#_Toc511752220)

[**2.3.** **Summary** 12](#_Toc511752221)

[3. Market Research 13](#_Toc511752222)

[**3.1.** **Reference to similar software** 13](#_Toc511752223)

[**3.2.** **Patterns and requirements** 15](#_Toc511752224)

[4. Artefact Design 16](#_Toc511752225)

[**4.1.** **Inventory Manager Design** 16](#_Toc511752226)

[**4.2.** **Data Access Layer Design** 18](#_Toc511752227)

[**4.3.** **Mobile Barcode Scanner Design** 20](#_Toc511752228)

[5. Artefact Development 20](#_Toc511752229)

[**5.1.** **Inventory Manager Development** 20](#_Toc511752230)

[**5.2.** **Mobile Barcode Scanner Development** 21](#_Toc511752231)

[6. Testing 22](#_Toc511752232)

[7. Evaluation 22](#_Toc511752233)

[8. Conclusions and Further Work 23](#_Toc511752234)

[9. References 23](#_Toc511752235)

[10. Appendices 25](#_Toc511752236)

Structure:

1. Introduction
2. Literature Review
3. Market Research
4. Artefact Design
   1. Inventory Manager Design
   2. Data Access
   3. Mobile Barcode Scanner Design
5. Development
   1. Inventory Manager Development
   2. Mobile Barcode Scanner Development
6. Testing
   1. Testing
   2. Types
   3. Results
7. Evaluation
8. Conclusions and Further Work

# Introduction

This report introduces the concept of an inventory management system implemented in

Java, and using machine learning to analyse item sales and predict future sales based on that data. Change slightly

The increased demand for optimal stock management in retail businesses has given software development companies the incentive for creating multiple different solutions to effectively store and manage data about a certain shop’s inventory. However, the current options available on the market usually come at a high cost, due to the specialised hardware included with them, and can, in some cases have hidden costs. Another problem which exists in most current inventory management software available is the discrepancy between the data about the items stored, and the actual items available, problem which has been tackled in the past with the use of RFID (Radio Frequency Identification) technology. The accuracy of today’s inventory management software is improving; however, it is not perfect. Machine learning models have been used in the past to deal with the perishable items in a store problem and created a precedent for using different machine learning algorithms for different purposes. The use of a mobile phone as a barcode scanner which sends the scanned data back to the main interface is not very common and doing so will reduce the costs by eliminating the need for specialised hardware scanners.

## **Aims and Objectives**

This paper proposes a Java based inventory control and management system, targeted at small/medium sized businesses. The main goal of the system is to make inventory control easier, and to minimise the impact of problems such as understocking or overstocking an inventory. Through the use of a mobile phone as a barcode scanner, the proposed system will ease inserting data into the inventory system based on item code readings, and by using two external JSON (JavaScript Object Notation) databases required information about the scanned items will be provided without it being hardcoded in the main database. One of the key features of the software will be a system that will read the data going in and out of the inventory, and based on what customers are buying most, make predictions as to what stock will be needed soon and in what quantities. This will reduce some of the work from an inventory administrator, who will no longer have to deal with ensuring the stock is always being maintained at optimal levels. Another important feature will consist of using a mobile application as a barcode scanner, allowing users to bypass purchasing specialised hardware scanners.

## **Techniques for realisation**

The functionality discussed above will be implemented using different tools, frameworks and algorithms. For data management the JDBC (Java Database Connectivity) API (Application Programming Interface) will be used, along with a web host service to provide data storage. The business logic will be separated from the application logic through the use of a DAL (Data Access Layer) which in simple terms will consist of DAOs (Data Access Objects) that will return data to the main application. This will ensure code readability as well as reusability, as a DAL can easily be changed to using a different database, or a set of multiple data sources without having to change the application logic along with it. The standalone mobile scanner application will be developed in Java as well, using Android Studio as the IDE (Integrated Development Environment). The mobile application will be developed using Android Studio, as a native Android application. This will allow both the mobile scanner app and the main interface to communicate easier, as both of them will be written in Java. The prediction system will be an implementation of the linear regression algorithm, which will take the independent variable as an average of the items exiting the inventory system, and the dependent variable as the needed quantities of items based on the independent variable.

## **Thesis Structure**

The remainder of the paper is structured as follows. Change slightly

Section 2 will discuss published research papers relevant to the context of the proposed artefact, and will analyse them, pointing out potential issues, or lessons to be learned which will help throughout the development of the Inventory Manager artefact.

Section 3 provides details on similar options available on the market and ensure that the functionality presented offers clear advantages for the potential users of this application.

Section 4 discusses and details the process of designing the artefact, describing the internal functionality as well as the desired outcomes. It also provides UML (Unified Modelling Language) diagrams of the functionality of the program, showing the thought process behind it.

Section 5 will detail the development process for all components of the artefact, in different stages. The problems that have arisen during the process will be recorded, and their solutions will be clearly described. This section also discusses the timeframe of the development, as well as the methodology followed during the creation process.

In Section 6, the testing approach is stated, and the testing results are shown in a table for ease of reading. change

Section 7 provides a critical analysis of all the aspects of the project, detailing the process behind creating the artefact. It contains the lessons learned while creating the Inventory Manager project and how it could be improved in the future. This section also compares the result with the set requirements and shows how they have been addressed and how some changed along the way. The evaluation section states potential features or design choices that could have been implemented and provides reasoning as to why they were not feasible or not fit for this project. The project management approach followed throughout the development is also discussed in this chapter.

Section 8 is the conclusion to the thesis. It goes over all the main aspects of the project, and summarises the main topics addressed by the thesis, reiterating the lessons learned from completing the artefact implementation. This chapter also discusses the possible impact the project might have on the current market, while providing a baseline for future work to be conducted by a similar project.

# Literature Review

## **Introduction to the Literature Review**

This section analyses the relevant work and research performed in the recent years, regarding the concepts of inventory management and stock control, and potential errors found in such systems. It provides a background in using machine learning techniques to improve the usability and accurately predict future sales, as well as studies on a dedicated reusable data access framework which separates the main classes of the program from the database. SQL (Structured Query Language), the standard language used to store and manage data in a database, is also researched in this section, along with the concept of a DAL (Data Access Layer) to separate business logic from application logic.

In recent years, studies about managing stock using more and more new technologies has seen a rapid growth, as the discrepancy between the physical stock of a company and the database records of said stock are being brought closer and closer together. Current studies address this discrepancy problem by bringing different ways of managing the inventory system so that both customer and retailer achieve maximum satisfaction. The studies discussed below provide the necessary base for developing new and efficient ways to minimise said discrepancy, via a different machine learning technique as well as a dedicated data access layer created in the Java programming language.

## **Relevant studies**

Currently, as all retail businesses use electronic systems to keep track and manage their stock, it is crucial that the software and hardware used are of high performance, and as low of a cost as possible, to maximise profits. Business systems are used to manage inventory based on a set of instructions, usually given by stakeholders of an organisation. This introduces a risk factor, as they are not always aligned to business goals and have the potential to cause errors.

Inventory management, as defined by Relph & Miller [[1]](#Relph1), is “the process of directing and administering the holding, moving and converting of raw materials through value-adding processes to deliver finished products to the customer”, and it “determines what levels of each products should be maintained, when stock should be replenished, and how large orders should be.” An inventory management system has the ability to keep track of and update stock data, using technologies such as RFID (Radio Frequency Identification), or scanning product barcodes to ensure that the number of items in the physical stock matches the one stored in its database. The management of inventory is supported by 3 concepts [[1]](#Relph1): planning, control, and balance. Planning refers to finding the optimal stock levels by sorting the items into categories as well as understanding customer demand patterns. By controlling the inventory, the system keeps track of stock integrity, meaning that it tracks the movement of items to accurately reflect where and in what quantities they are stored. Balancing means making sure that the inventory is moving according to plan, by looking at the supply and demand factors which point to what is needed and what is being sold.

Relph & Miller [[1]](#Relph1) describe material resource planning (MRP) systems as being used to manage material planning based on different values which need to be taken into consideration:

* Quantity of items required, and timeframes between which the inventory needs to be restocked. This variable is difficult to estimate, as customer demand can be uncertain at times; restocking the inventory also depends on the customer demand, which means that a solution to accurately predict this demand is needed.
* Potential errors in the supply/demand chain.
* Amount of time taken for new items to reach the stock.

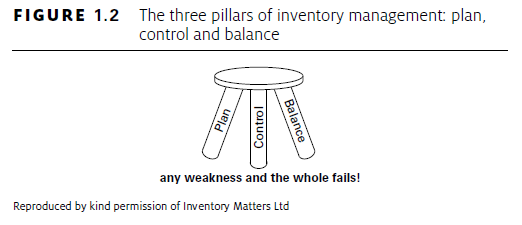


Figure 1-Source: G. Relph & C. Miller, Inventory Management - Advanced methods for managing inventory within business systems

Inventory systems are designed with the goal of preventing against both under stock and overstock, as understock leads to customer orders not being fulfilled on time or lost all together, and therefore losing profit; overstock causes money to be unnecessarily spent on purchasing stock that’s not needed, as well as managing that stock, and therefore causes a loss for the company. Overstocking also causes issues with storage space, as the space where the unnecessary items are stored is taken up and unable to be used for the normally rotating products.

Relph & Miller conclude that translating the financial inventory needs of a business into item-level instructions is essential for any such system, as this way the inventory is well-managed and optimum stock levels are achieved.

As mentioned above, certain errors can occur when storing information about stock in a classic inventory system. One of the main challenges with most existing inventory management systems and point-of-sale (POS) services, as pointed out by Selma Khader et al. [[2]](#Khader2) is the usual discrepancy between the quantity shown in the information system (i.e. database) and the physical stock. The ideal inventory model assumes that the quantity shown in the information system (IS) is the same as the actual number of items on shelves, however that is not the case for most systems. Different variables can create a discrepancy between the two, known as inventory inaccuracy, which can be classified as follows [[2]](#Khader2):

* Inaccuracies due to shrinkage – products can be deteriorated or stolen without being detected by the IS and as such cause errors
* Temporary discrepancies – this is caused by misplacement of the items, i.e. when a portion of the inventory is stored in the wrong location
* Transaction and record errors – impact the IS inventory while the Physical Stock remains unchanged (scanning the wrong item, adding the wrong number of items as exiting the inventory [for example pressing 5 instead of 2 items at a cash register, or variations of the same item being grouped all together and as such only one of the groups gets updated]
* Supply yield – the supplier could accidentally supply less or more than the requested quantity and as such the IS would not detect any changes

Atali et al [[3]](#Atali3) believe that the main cause of inventory discrepancy is shrinkage, the cost of which is a major concern. They define “Phantom stockouts” as missing inventory due to it being stored in places not accessible to customers. Ineffective inventory management leading to phantom stockouts is an important factor to consider when taking into consideration the effectiveness of any powerful inventory control system, as that can cause a significant loss in sales. The authors argue that “The effectiveness of all the powerful inventory control systems that one can design would suffer when a discrepancy problem exists. It is therefore important to develop inventory control systems that factor in consideration the unobservable inventory discrepancy. “

Such errors have the potential to greatly impact the profits of a business, as the discrepancy discussed above can cause overstock and understock problems, which would lead to the business not meeting customer demands, and therefore loss of profit. By creating a system which can predict future customer demand, the impact of such errors can be reduced, and by making sure that each item is scanned individually as it exits the inventory the understock type errors can be prevented.

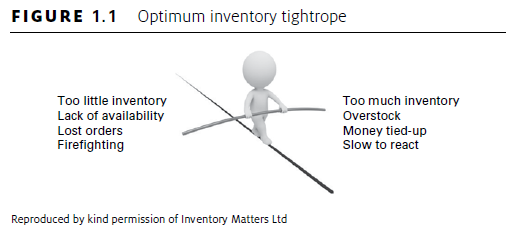


Figure 2-Source: G. Relph & C. Miller, Inventory Management - Advanced methods for managing inventory within business systems

In order to efficiently manage the data about inventory, some form of a data storage/management system is required. SQL (Structured Query Language) is one of the most widely-used languages for managing data held in a DBMS (Database Management System), and one of the most versatile. The execution of SQL instructions in the Java environment is allowed through the JDBC API (Java Database Connectivity - Application Programming Interface), which “allows the call to SQL queries to any DBMS” [[4]](#Lungu4). In Java’s client-server architecture, the databases can exist either on the same machine (local databases) or on a different one, located somewhere else to which the client connects via a network. The results are independently and consistently processed, which allows developers to create robust and scalable applications using JDBC. Lungu & Mercioiu [[4]](#Lungu4) agree that JDBC can be used to access databases in an object-oriented way, using classes and interfaces which cover abstract concepts. It also created metadata by defining interfaces which give developers important information regarding the database queried, the DBMS used, etc. The authors conclude that using the JDBC API to connect to SQL databases allows for simple and time-efficient ways to store and manage data, and perform CRUD (Create, Update, Retrieve, Delete) operations on a DBMS.

To allow code reusability and provide an interface from the application to the data, it is good practice for developers to include a separate layer in the code, dedicated to accessing the data in an efficient way, and separate the business logic code from the system infrastructure code [[5].](#Seaman5) This is known as a DAL (Data Access Layer) or Data Access Framework. Minore et al [[6]](#Minore6) propose a Data Access Layer as a system that groups together database commands in a common reusable data access component, as well as encapsulate data access logic. They describe the advantages of using such a layer as “improving developer productivity and/or reduce the quantity of runtime errors introduced into the code.” In their patent from 2013, Chatterjee et al [[7]](#Chatterjee7) introduce an improved version of the DAL architecture by using drivers installed on Web servers which communicate with the DAL servers so as to improve scalability and use system resources more efficiently. The proposed method of implementing the discussed architecture would consist of dedicated “DAL servers which present a virtual database to the web servers, (…) [which] in turn open connection to a set of physical databases.”

This framework fits into the context of the proposed artefact, as it creates an easier and more efficient way to organise code, and as such the software performance as well as code readability would be improved. In the context of the object-oriented environment provided by Java, the DAL would be comprised of multiple DAOs (Data Access Objects) which are defined by Seaman et al [[5]](#Seaman5) as “an abstraction to a data source, such as a database”. The code is usually split into multiple DAOs, each being related to one entity (or table) so as to facilitate maintenance, and to organise the queries to the database according to the respective tables being accessed. As each entity in a system needs its own DAO, the developer has then to decide in which object each query should go, which according to Seaman et al [[5]](#Seaman5) can be a potentially difficult task, especially when the goal is to improve readability of the DAOs. This is due to queries that touch different tables, using the SQL JOIN command, which can produce ambiguous results in the Java code. This should not be the case with the database used for the proposed inventory control system, as there will not be a large amount of data, and in turn it will be stored in a few tables at most.

To predict future sales, as mentioned in the introduction section of this paper, the linear regression function needs to be defined and used. Linear regression is defined by Tantawi [[8]](#Tantawi8) as “a method of estimating the relationship between two or more variables based on observed data”. That relationship can be expressed as a linear equation, in the form of , and the line defined by it is called a regression line, or line of best fit. When the data resulted is displayed in a scatter diagram, the results roughly follow that line. There are two types of linear regression, one that takes a single variable as an input and returns an output, known as simple linear regression and one that takes two input variables and returns an output based on both, known as multiple linear regression. In simple regression, the value of the dependent variable *y* changes in response to the independent variable *x*, and the regression line is represented by the above-mentioned equation, where *a* is the value of y when the line crosses its axis, and *b* is the slope of the line.

The proposed artefact will implement simple/multiple linear regression as an algorithm in Java, having the *x* variable as the average quantity of an item category sold in a certain amount of time, and *y* as the dependent variable, which will represent the stock needed. The data required for the algorithm will be retrieved from the database and processed, and the results will be displayed in the form of a prompt to the user, suggesting the quantity of each product that needs to be ordered.

Due to the increase in availability of machine-readable data, operations management has new opportunities for better decision making. In the context of inventory management, demand is the key uncertainty affecting decisions, and as such large-scale web data can be used to improve inventory decisions. Bertsimas et al [[9]](#Bertsimas9) study how big data can correctly and successfully inform inventory management decisions by analysing a case study of one of the largest multi-media distributor in the world. They define the problem as the vendor’s need to maximize network-wide sell-through with limiting factors such as limited storage capacity, and the uncertainty in the initial demand for new products. By analysing the company’s internal data, which consists of over 4 years of sales, they construct inventory prescriptions which are then further analysed to reveal a result of 88% accuracy in predicting sell-through volumes over a span of 150 weeks.

Large amounts of data can therefore be used to accurately predict future sales, as well as customer demand, which in the context of the proposed artefact, means that linear regression is able to predict the future stock needs, and as such is a powerful tool in efficiently managing inventory.

Companies producing perishable products need to consider the age of the products as well as the quantity when managing their stock. That is due to items such as dairy products, fresh meat products or pharmaceuticals having a very limited shelf time, which if not managed properly can reduce the inventory of the respective company.

Ahmet and Ibrahim [[10]](#Ahmet10) propose age-based replenishment policies which take into consideration quantity as well as age of the stock, and use reinforcement learning methods to solve the problem of perishable items in relation to uncertain customer demand. They used reinforced learning to show how age information affects the perishable inventory system by applying the Q-learning and Sarsa algorithms and aimed to obtain a near-optimal replenishment policy for a perishable product. They conducted the cost analysis taking into consideration different variables, including lead time, cost ratio and product lifetime. The results concluded that policies which considered both age information and inventory quantity gave more optimal results in contrast to the policies which depended on quantity alone. Furthermore, they concluded that the Sarsa algorithm is better when it comes to cost performance, however, the Q-learning model for the base policy pointed to better results in contrast to the Sarsa algorithm. The paper also analysed how the discount factor, learning and exploration rates affect the performance of the regression learning model used, and showed how the initial-decay values of both exploration rate and learning rate affected the learning model.

Therefore, Ahmet & Ibrahim’s research further expands the concept of using certain machine learning paradigms in the context of an inventory management system and provides basis for further development in this field.

By using supervised learning, more specifically the linear regression algorithm as proposed by this paper, the artefact discussed has the potential of minimising the discrepancies between the physical stock and the information about the stock stored in the system. This solution will ensure that the artefact will prove to be useful to stakeholders who wish to improve their stock control, and as such it will be marketable.

## **Summary**

Past and current research indicates that accurate inventory tracking and stock control is a problem that affects many business owners. As such, different solutions and methods for solving that problem have been discussed and analysed, along with technologies which could potentially help create a system that could minimise profit losses of stakeholders due to inventory discrepancies. The literature discussed above helps in pointing out the main issues with current stock control systems, provides details about the platform used for developing the proposed solution, describes technologies used for creating that solution, and provides previous data about using machine learning techniques in the context of inventory management.

# Market Research

In order to better understand the user requirements and ensure product marketability, the current market needs to be analysed, so as to learn what the current options available provide, and what they are lacking.

## **Reference to similar software**

Finding out who the users of the system will be is one of the most important parts of conducting market research, as they will provide the requirements for building the software. As with any other inventory management software, the target users will be certain business owners/managers, particularly those owning small and medium sized inventories.

By looking at similar options available on the market, the product presented by this paper can be fitted into a category, and a comparative analysis of it versus competitors can be performed.

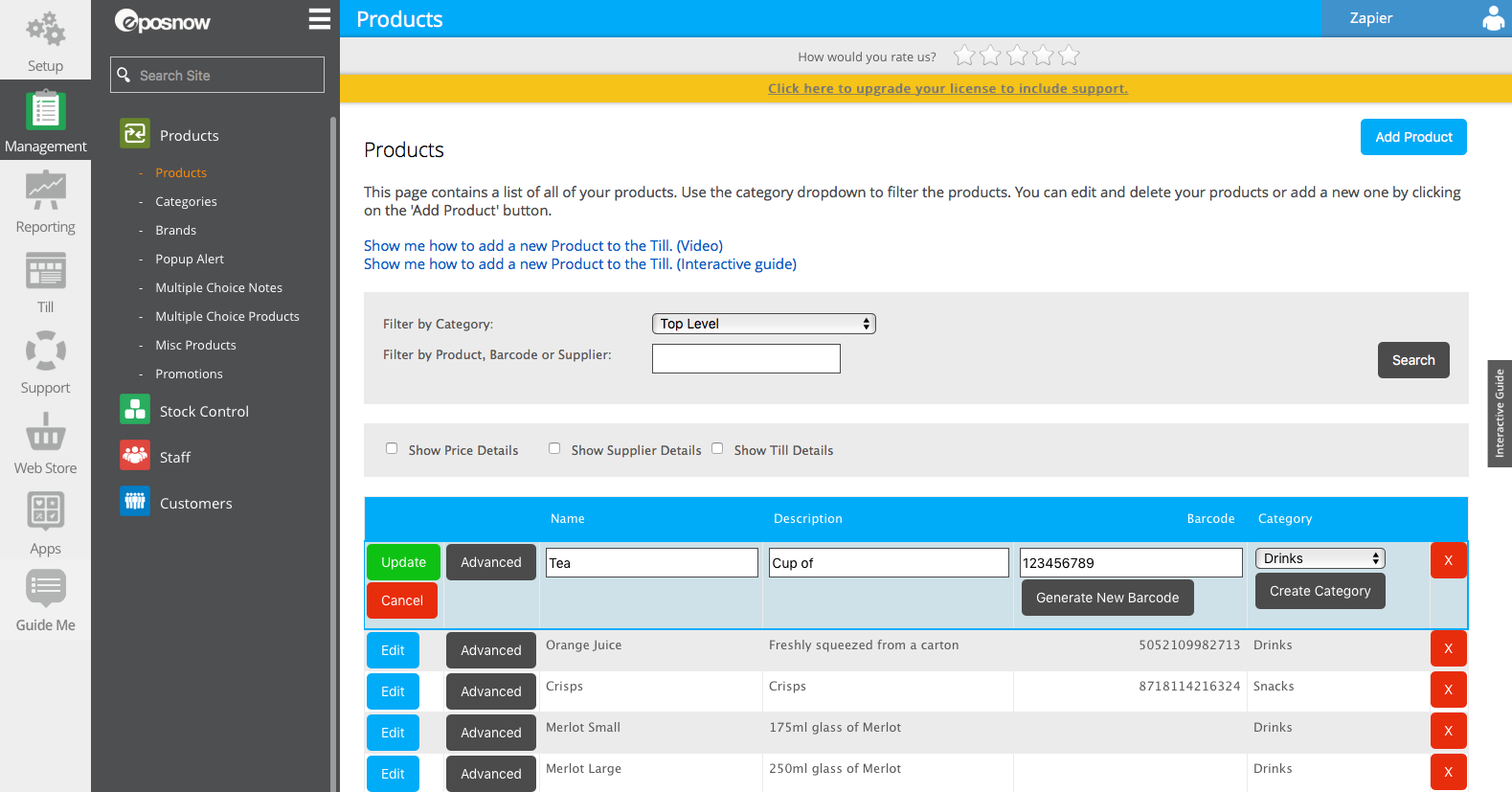
EPOS Now is one of the largest POS (Point of Sale) and inventory management systems available, winning a Queen’s Award for Enterprise in 2016. According to their web page [[11]](#EPOS11) their software is used by over 30.000 businesses, and allows access to the system from anywhere, monitoring and reporting in real time, as well as connecting to other available software such as payment systems, online ordering platforms, etc. They offer solutions with specialised hardware, Android and iPad POS systems, as well as standalone software with monthly or upfront payment. The cost varies a lot, depending on the plan purchased, for example a retail POS complete with hardware costs £1200, while the standalone software can be acquisitioned for as little as £10 a month.

Figure 3-EPOS Now Inventory Manager

However, according to customer reviews from [[12](#TrustPilot12),[13](#SoftwareAdv13)], this system is not perfect, and certain features are implemented poorly. There are hidden costs, for instance access to tech support is an extra £30 a month on top of the licence price, and some users have said they were charged this money without signing up for it. The claims of EPOS that it fully integrates with a system such as SageOne for accounting have been tested by a customer, and proven wrong, as the system does not sync product data between the two.

Wasp Barcode [[14]](#Wasp14) is another provider of inventory management software, as well as specialised item tracking hardware. They provide different business plans, priced differently, and product management, as well as offering assets tracking to minimise stock losses. Along with that, their software offers a notification system which alerts the user of low-level or near-expiry date stock and barcode design for labelling products. While not as widely used as EPOS Now for example, it still offers a large number of features, however the pricing is higher. The Standard Package offers 1-user inventory control system along with specialised hardware (barcode scanner and receipt printer) for prices varying between £950 and £1600, which can go up to £3800 for the Enterprise editions of the same product. While they do provide an option to purchase standalone software, they do not offer a monthly purchase plan, selling the program for prices ranging between £500-£2200. The high prices could prove to be a problem for a new business owner who has to deal with a lot of other costs and might not want to spend that much on inventory management tools.

## **Patterns and requirements**

As it can be seen from the products presented, the pattern following the marketability of the software/hardware revolves around licensing, and different priced packages for different businesses. This is not a bad strategy, as it allows the product vendors to offer different types of service to their variety of users, but it can happen that a certain user might not know about some of the costs. While that may not necessarily be a problem, charging a fixed price for a one-time purchase of the software seems to be beneficial to the buyer, as it provides transparency to the transaction. Most of the current inventory systems available provide different functionality in one package, as well as documentation on how to get started with using the system, but in some cases the initial setup of the software and hardware infrastructure can take a long time. The proposed solution takes into consideration the above-mentioned concerns and addresses the problem by offering a simple software which can run on any Windows machine, as well as an Android application and a web page, which do not require too specialised hardware to function. As such, the overall usability of the system will be boosted, and the user will not have to spend too much time setting up the program.

Taking into consideration the applications presented above, a list of requirements can be provided. One of the main concerns about current available options on the market is high cost. The issue will be addressed by allowing the buyer to pay a fixed price for a one-time purchase, without overwhelmingly complicated package options with different features. Users will be presented with the full functionality of the system and the option to pay only after the first month of use, in order to allow them to decide if the software is up to standards. Along with that, the prediction model implemented needs to be highly accurate, as that will provide an incentive for the users to purchase the program.

# Artefact Design

This section discusses the complete artefact design patterns, providing UML diagrams for describing the inner workings of the software. It is further divided into three sub-sections: Inventory Manager Design, which will discuss the implementation of the main Java-based software; Data Access Layer Design, in which the concept of a Data Access Layer is described, and its functionality integrated into the artefact; and Mobile Barcode Scanner Design, which shows the thought process behind the mobile application used along with this software.

## **Inventory Manager Design**

By analysing the requirements found in the Market Research section, designing the Inventory Manager artefact was a relatively straight-forward process, with only a few minor adjustments made later in development.

As the programming language used was Java, and the IDE of choice was Eclipse, designing the user interface was aided by Eclipse’s Window Builder plugin [15]. The plugin offers a drag and drop feature for positioning UI elements, as well as automatically generating the necessary code in the background.

A draft of the user interface was created before the development started, using drawing tools to approximate the position of certain elements on the screen and to provide an idea as to how the program should look like. The original draft was promptly created in Eclipse, and more components were added to it as required.

[insert image of original layout/login screen or something]

The user interface consists of [¾] JPanels, which act as different screens the user sees. The panels are shown and hidden on button presses, creating the transition from one screen to another.

-sockets implemented in the main method

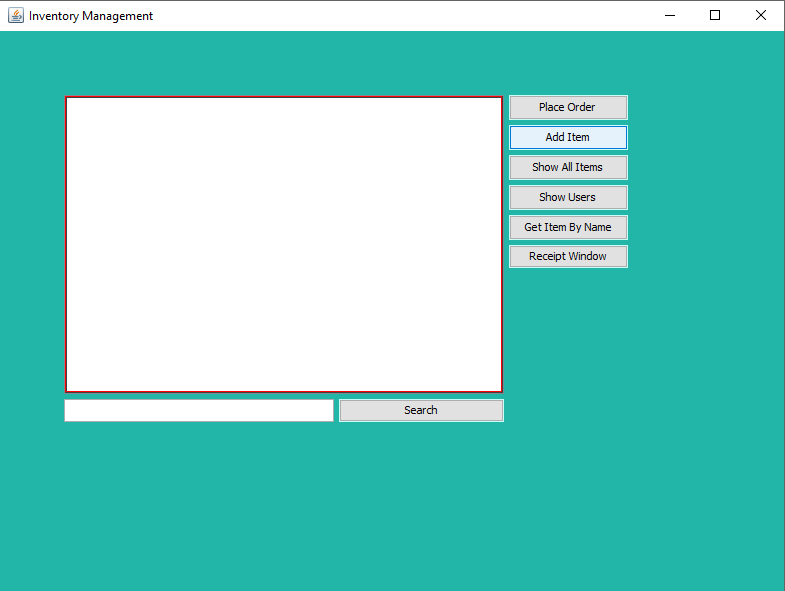
-discuss class organisation – 3 packages

-how the interface accesses data

-changes to internal structure during development: data access methods fitted to accommodate item scanning and adding to db

-regression class is separated from the main program as it requires a lot of code which would clutter

-discuss cluttered code, provide code samples, code is commented accordingly



## **Data Access Layer Design**

To allow code reusability and provide an interface from the application to the data, it is good practice for developers to include a separate layer in the code, dedicated to accessing the data in an efficient way, and separate the business logic code from the system infrastructure code. This is known as a Data Access Layer (DAL) (Seaman, Nord, Kruchten, Ozkaya, 2015). The DAL connects to the database using a helper class, which creates and manages the physical connection. SQL statements are executed in the DAL, and the data is returned in the form of DAOs (Data Access Objects) which are used by the application layer.

-connection pooling

-apache libraries

-server database, later changed to local database – mention why

-do not use db drivers to connect to internet servers

-data access objects – users, items, orders

-orders never used – potential future feature

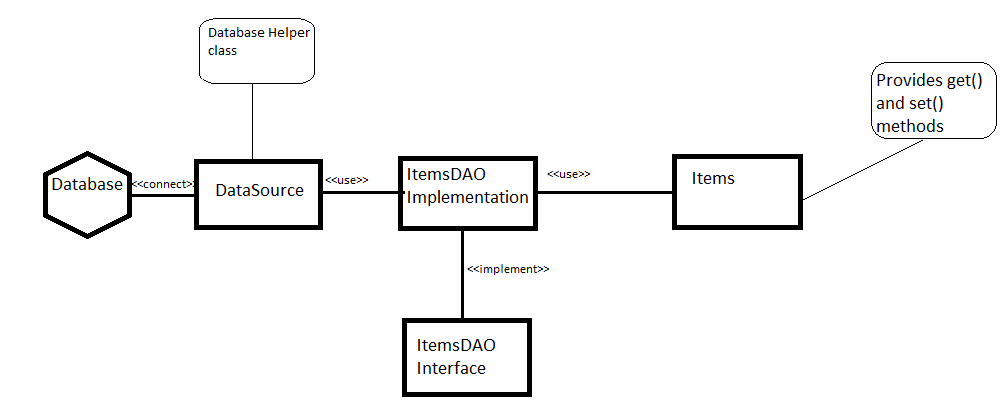
-discuss DAL pattern – class diagram ready, maybe make a simple version to put here and move the big one to appendix

-interfaces are implemented by the actual classes, there is a get/set class which is used by the main UI

-how the objects are accessed

-Try with resources block for some

Below is a simple diagram showing the DAL structure, more specifically the implementation of the Items object. A more detailed class diagram showing all three DAOs in the system can be found in Appendix 2.



The DataSource class creates and maintains the physical connection to the database, while also providing a “pooling” feature, meaning that the class does not dispose of the opened connections as they are closed, rather keeping them cached so that they can be instantly accessed later as they are needed. This process speeds up the database transactions and contributes towards saving resources on the user’s machine. The DataSource class returns the connection to the ItemsDAOImplementation class, which is where the SQL statements are executed, and the results returned are kept. This class is an implementation of the ItemsDAO interface, which is what the Application Layer uses to retrieve the required data objects. The Items class provides getter and setter methods for the different attributes of the respective object. Concurrently, these classes work together to retrieve data from an external database and return it to the application layer in an efficient way.

Advantages of using a DAL include:

* Abstracting the data store so that the data source can easily be changed if needed
* Code separation
* Provide common calls to retrieve objects (such as a getAllItems() method)
* Once a DAO has been created the general layout can be repeated for others, simplifying coding

Disadvantages:

* The more DAOs there are in a system, the more tedious coding gets
* It can be quite complex to build

Connection pooling was implemented using a suite of libraries provided by Apache, called commons-pool.(Apache Commons, 2017), which consists of three classes referenced by the main Java program.

## **Mobile Barcode Scanner Design**

-use xml to design user interface

-describe layout, screenshots

-discuss how google vision is different from zxing and why I chose to change

-how the external libraries are referenced: gradle

-main activities of the app- what goes where

-async task for sockets

-why async

-async task for json retrieval

-class diagram or use case diagram or activity diagram

# Artefact Development

This section discusses the implementation process of the Inventory Manager artefact, detailing the features implemented, problems and their solutions, as well as discussing the timeframe of the project with reference to the GANTT chart created before the start of the development phase. Any deviation from the chart will be given reasoning, and changes that have occurred throughout this phase are described, as well as the impact they had on the overall success of the project. Along with that, the developer’s approach regarding project management, and the methodology followed is also mentioned in this chapter. The remainder of this section is divided into two sub-sections, each comprising an element of the final artefact. Section 5.1 will discuss the development of the main Java program, and section 5.2 will detail the process of creating the Android Mobile Barcode Scanner application.

## **Inventory Manager Development**

The development of the project began soon after the initial UI drafts were completed. The first step was to register with a free mySQL server host, so as to create a database to be used by the program. Once the desired structure of the data was created, and the dataset was normalised, the next step was creating the connection from the Java program to the server. Using the server details provided, the main class of the program was able to create and maintain a connection over TCP/IP, however that was only for testing purposes. The next step involved creating a database helper class to manage the connection separate from the main UI threads, and to separate the application layer from the back-end code. The class was implemented using connection pooling, which reduces the number of times a connection has to be opened for database transactions. Connection pooling [16] uses a “pooler” class to keep ownership of the physical connections and manage them, and when a connection is closed it is returned to the pool to be reused later. In the Inventory Manager artefact, connection pooling was implemented using a group of external libraries provided by Apache [17] (commons-pool) which sped up the development process. The database helper class does not return the opened connection to the main UI thread, but rather to a Data Access Layer consisting of three different DAO’s (Data Access Objects) which contribute to separating the application layers. As discussed in section 4.2, the Data Access Layer …[discuss DAL]

-maybe separate into firther subsections – like DAL, Sockets, etc

After having a DAL in place for performing database operations, the

-layout changes – use cardLayout

-implement login method – password not hashed

-refine login method

-implement sockets to get message from android app

-automatic login by scanning the login barcode

-implement manual adding items to the db on a separate screen

-items are added to db automatically when scanning a product

-items already existing in the db are updated quantity rather than adding new ones

-if the receipt screen is visible then the items are not added but rather removed

-use Apache maths to perform regression

-scan a product and show information about how much has been sold and how much will be sold

-summary

Before the start of the development phase, a schedule was set, using a GANTT Chart to

## **Mobile Barcode Scanner Development**

Maybe separate into firther subsections, like Sockets, Scanner library, JSON

-introduction to the scanner – platform, language, and features to be implemented

-tools and frameworks to be used

-first draft of the scanner app – only scan the barcode and return the number as text

-layout changes, updated design – filan

-switch from using google vision to zxing – flashlight problems

-app scans and returns to the main screen – no continuous scanning

-implement sockets in android – async task which sends a message to socket

-look up various databases with EAN barcodes to get product information

- (mention outpan api which didn’t work because its no longer maintained)

-upcitemdb.com to get product data as JSON

-create JSON object retrieval class in android

-problems with the format – returned data is an array and my class couldn’t handle that

-attempt to implement continuous scanning mode – fail

-fix the array issue – look through code to find out how

-add a second api access, second website – JSONResult4

-make it so that the data gotten from json classes is sent to sockets as well

-continuous scanning mode works

# Testing

-create custom dataset for purchases

-run the regression algorithm on the dataset

-past data is shown to test accuracy

# Evaluation

-lessons learned from creating the complete project

-how it could be improved

-features that I chose not to implement

-why I chose external libraries for my project rather than coding shit myself

-reiterate requirements and show how the project meets them

-project management approach

-methodology followed

-time management

# Conclusions and Further Work

-reiterate lessons learned from project

-go over al lthe main aspects

-discuss impact the project might have – how not many other options available provide item sales prediction

-propose features that could be implemented in the future or by a different project – baesyan probabilities and other algorithms, database server, etc

# References

[1] G. Relph and C. Miller, **“Introduction to Inventory Management”** in **“Inventory management: advanced methods for managing inventory within business systems”**, Kogan Page, 2015

[2] S. Khader, Y. Rekik, V. Botta-Genoulaz and J. Campagne, **“Inventory management subject to multiplicative inaccuracies”**, *International Journal of Production Research Vol 52, iss. 17, 2014*)

[3] A. Atali, H. Lee, O. Ozer, **“If the Inventory Manager Knew:**

**Value of Visibility and RFID under Imperfect Inventory Information”,** Stanford University, September 2006

[4] I. Lungu, N. Mercioiu, “**Database Access Through Java Technologies”,** *Database Systems Journal, vol. 1, Jan. 2010*

[5] C. Seaman, R. L. Nord, P. Kruchten, I. Ozkaya, "Technical Debt", *ACM SIGSOFT Software Engineering Notes*, vol. 40, pp. 32, 2015

[6] L. Minore, P. Yuknewicz, S. Lasker, A. Cote, **“Data access layer design and code generation”**, U.S. Patent 7 711 740, May 4, 2010

[7] A. Chatterjee, C. Shah, M. Kishore, J. Hsu, **“Data access layer”,** U.S. Patent 8 589 365, Nov. 19, 2013

[8] R. Tantawi, **“Linear Regression”,** *Research Starter Topics,* Research Starters Database, 2013

[9] D. Bertsimas, N. Kallus, A. Hussain, “**Inventory Management in the Era of Big Data”,** *Production & Operations Management. Dec2016, Vol. 25, Issue 12*

[10] K. Ahmet, D. Ibrahim, **“Reinforcement learning approaches for specifying ordering policies of perishable inventory systems”,** *Expert Systems with Applications, Jan.2018, Vol. 91*

[11] EPOS Now Point-of Sale System, available online at <https://www.eposnow.com/uk>, accessed on 14/12/17

[12] TrustPilot UK – Customer service reviews, available online at <https://uk.trustpilot.com/review/eposnow.com>, accessed on 10/01/18

[13] SoftwareAdvice – Reviews, Pricing and Demo, available online at <https://www.softwareadvice.com/uk/retail/epos-now-profile/>, accessed on 10/01/18

[14] Wasp Barcode stock control systems manufacturer, available online at <https://www.waspbarcode.co.uk>, accessed on 28/12/17

[15] Eclipse Window Builder, available online at <https://www.eclipse.org/windowbuilder/>, accessed on 03/12/2017.

[16] Connection pooling, available online at <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql-server-connection-pooling>, accessed on 10/04/2018

APACHE COMMONS POOL <https://commons.apache.org/proper/commons-pool/>

# Appendices

Appendix 1: Poster

Appendix 2 – DAL Class Diagram

