Bachelor's thesis

Degree programme in Information Technology

2018

Luca Zelioli

AUTOMATED INVENTORY APPLICATION

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology

2018 | 70

Luca Zelioli

AUTOMATED INVENTORY APPLICATION

The aim of this thesis was to create an automated inventory application to reduce the working hours and to minimize errors in the inventory process of a small enterprise. The application was done by using a Microsoft C# programming language. The goal was to combine business intelligence with computer engineering.

The theoretical background demonstrates the language and framework used to build the application. Also a description about the connection with a GSM modem and the computer application is given. The practical part of this thesis describes the process used to create the application on the client and server side including the realization of graphical user interface. In addition, a deep view of the functions and the process of coding is given.

The project was divided in two branches, the server side and the client side. The first one collects the inventory data used by the management, and the second one is used by the sales agents who send the data to the server side with an Android application or with SMS messages. The results show that the automated inventory application presented has been a successful implementation for both sides. The given feedback has been positive. The commissioner of this thesis has confirmed that the application has been a useful and essential update to change the old inventory process.

KEYWORDS:

Inventory, business intelligence, SMS message, computer engineering, automated application.

I deeply thank the Professor Tiina Ferm, Professor Poppy Skarli for their availability and their advice.

I thank the company Lappi-Hunaja for the help, the kindness and the professionalism that was granted to me.

For their support and encouragement my sincerest thanks to my family.

I thank my best friends in Santa Caterina Valfurva, my second family, for the wonderful adventures experienced together.

Luca Zelioli

CONTENTS

APPENDICES

PICTURES

LIST OF ABBREVIATIONS (OR) SYMBOLS

C#	Computer programming language developed by Microsoft in 2000
CLR	Common Language Runtime
COM	Communication
CPU	Central Processing Unit
GPS	Global Position System
GSM	Global System of Mobile communication
GPRS	General Packet Radio System
IDE	Integrated Development Environment a software that is used to create other software
MSIL	Microsoft Intermediate Language
SQL	Structured Query Language a common language used in Database management.
WPF	Windows Presentation Fundation type of Windows Application

1 INTRODUCTION

Lappi-Hunaja is a small company, situated in Lappi, which is a small village in Rauma in the Satakunta province in South West Finland. Over the years, this family business, has become one of the most well known honey producer companies of Finland. The main business is to sell rare honey, in several Christmas markets in Germany. Untill three years ago, the selling process was run smoothly as the number of the markets were limited to under thirty. However, the situation changed, when the number of places grew to thirty-five.

An inventory could be defined as the management of stock inside a company. It can be used as a means of control of how many products are stored in the company warehouse, how many products are sold, and consequently predicting which products, it is necessary to focus on in the future. This management process is important in small companies such as the one analysed in this work.

Usually the production assets of this type of enterprise are not huge and must be piloted and, in this case a good inventory system could make the difference. Receiving regularly the sale stocks helps to calibrate the production, because if it is decided to produce small amounts of products, there is the risk that there are not enough goods to sell, and vice-versa.

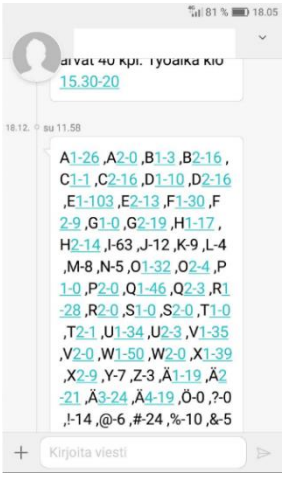
The small company in this thesis, has a crucial objective to always have a good sale feedback because honey, the main-stream product of the small company analysed here, has an expiration date. The current method of inventory has became obsolete, with two mains problems: the first one is the human error in updating the inventory. The second problem is evident when, at the end of the inventory process, it is hard to correctly calculate the real revenue if the sales figures are not accurate in the first place. The solution of this problem is to develop a new application for this company.

The automated inventory application aims to improve the business intelligence and, increase the circulation of information. The system, operates receiving sales information, from 35 different places and sales agents. The information comes in the form of SMS. When the message arrives, the program retrieves it and it stores the obtained data in the database. The Automated Inventory Application could be defined as a collection of information that is organized so that can be easily accessed, managed and updated. The

data is organized in a manner that it is easy to find what is needed. In the modern relational databases, the data could be accessed in different ways: by interacting with a database, by being a database user, and by using the structured query language [1].

The inventory operations, twice a week, have always been slow, developed with an old template, the previous procedure is as follows: the various sales agents, after counting the final honey stock, they fill in a prepared text message, ordering the quantities they need, with alphanumeric symbols instead of products names (Picture 1), by adding the number near the symbol. At this point, the message is sent back in Finland, to the author of this thesis, and these numbers are entered in an Excel file. This operation is time consuming and prone to mistakes, and it is summarized as follow:

- Count Stock.
- Fill the text message.
- Send SMS.
- Collect inventory items manually into Excel file.
- Send Excel file to the Management of the company.
- Management decides which products to send based on this file.
- Products are delivered.



Picture 1. Example of SMS received.

When the Excel file is completed, it is sent to Germany where a manager prepares the physical honey products for the transportation to the various points of sales.

This system could be defined as an improvement to the traditional slow method that the the commissioner had used until now, with the certainty of minimizing human error at this stage, increasing the time spent in other tasks.

The purpose of this thesis, is to create a software that could receive text messages from mobile phones, check and process the information in these message with C#, and put it in a database. This software can generate important information for the business intelligence, such as sales projection, and revenue generated per day. It can regulate the production of honey during the Christmas period, a crucial operation because the production is based on the sold. It is beneficial for the logistics department because the time wasted in waiting could be spent in driving at earlier hours. Making the logistics operations more efficient improving the overall quality of the company operations.

An important improvement is the sending system, instead of using a normal text message from the phone, an application is developed, with all the important information that sometimes the sales agent forgets to include as their name and the name of the point of sales.

Thanks to modern techonolgy, the inventory is sent via message service, using a traditional GSM modem. The Application handle errors, making for example connection test between the network and itself, or between the database and the software.

The project is composed by a “centralized application” built as Microsoft Windows Presentation Foundation, defined as a computer programming model which permits the compilation of modern application. The programming language used is Microsoft C#, a powerful modern language developed in 2000.

This project used Microsoft Visual Studio Professional 2017. The system need to be connected to the cellular network.

In this thesis, after a short theoretical background about this subject (Chapter 2), and in Chapter 3 gives a precise description of the Automated Inventory and how it works. Chapter 4 also described how the Android application is built and how it makes the inventory process simpler for final users. In Chapter 5, the server side of the Automated Inventory is analysed and explained. In Chapter 6, the operational challenge are taken in consideration. In Chapter 7, the evaluation of results are proposed, followed by Conclusions in Chapter 8.

2 THEORETICAL BACKGROUND

In this chapter, a theoretical background of the Automated Inventory Application is given. A brief introduction about the .NET Framework and its components is described. This is followed by a description of the C# Programming language with emphasis on the class used to develop this work. In the last part, this chapter gives a description of a modem, the GSM network, the GPRS.

2.1 Development platform

The .NET Framework is a development platform, and it provides a managed execution environment for Windows, creating services when running an application. One of its components is the CLR, i.e., common language runtime, the workspace where the application run, and the .NET Framework Class Library that provides a library to improve the development of code. The CLR works directly with object and its references, also called managed data. This procedure helps to eliminate memory overloading. Even objects written with different programming languages could operate with each other inside the CLR. This is possible because languages share the common type defined inside this component.

The major characteristic of the .NET Framework platform could be summarized as follow:

- Memory management: it provides in autonomy the memory allocation for running the application.
- The basic types are defined directly by the .NET Framework and is common to all type of language.
- Over 4000 classes organized in namespace with various functionalities that helps programmer to develop modern application.
- Routines written in one language are accessible to other languages.

The .NET Framework provides the CLR, a class libraries that help a developer to build robust software, it includes also the runtime environment with the main task to runs the code and make the development process easier [2][3].

2.2 C# as development language

C# is programming language developed by Microsoft in 2000. It is composed by one or more source files, an ordered sequence of instructions. It has functionality such as the .NET framework that helps in building elaborate application from different computer field, for example for networking application and database plug-in for different usage. C# programming language is used to build several types of applications in all major field, for example Web service, client server application, and more. C# is a language with simplified syntax compared with C and C++. It supports generic methods and type that increase the safety and the performance of it. It is an object-oriented language and it is possible to use programming techniques:

- Encapsulation: that a group of properties of an object stays inside that object.
- Inheritance: the possibility to create a daughter class that has a direct relation with the parent class.
- Polymorphism: interchangeability between multiples classes.
- Delegate: method signatures, to enable or invoke functions.

A class is a data structure that contains members such as methods, properties, events, operators. It is a tool for creating new types used as built-in types. It is possible to make relations between classes, and take advantages from it. The properties for classes are summarized as follows:

- A class is defined by a developer.
- A class has members.

A class declaration consists in a set of attributes followed by an optional set of class modifiers (for example, public, private, protected, etc.) the keyword class and the name of the class. The body is closed in curly brackets. Some classes need explicit declaration to be initialized, the function that allows this procedure is called constructor. It is possible to recognize it because it has the same name of the class, in this case, all the objects are initialized by the construction call [4][5][6].

An object is a block of memory that the compiler allocates and configure according the specifies of described in the class. Objects are also called instances and it is possible to store its references inside a variable object [7].

C# uses Xamarin platform to build Application that interact with the Android Operating System. The API include all the technologies used normally to develop Android Application as Objective-C, Swift and Java. C# avoid verbose type annotation using type inference. The applications are responsive thanks to the Asynchronous programming [8].

If an application needs to interact with other Microsoft Windows software and, with the COM object, it uses a process called Interop. The COM object makes the work easier to use the application inside the Office software packet. Interop is used for example to enable the communication between C# and Microsoft Excel, using the Microsoft Office Interop Excel namespace, where it presents important tools such as definition of cell range, worksheet and similar [9][10].

C# uses the System.IO.Port namespace to communicate between the program and external devices attached with the USB cable. This namespace contains classes to control the serial ports. When the connection is established, the communication takes place using the AT command set defined as a set of instruction to control a modem. The commands are always sent with the acronym AT followed by a set of strings, when return button is pressed the command is sent to the modem. The commands can be delivered as basic mode, single command, or extended mode that is a combination of the basic ones. These commands are able to manipulate the register, the short memory of the modem, for example to delete SMS from the modem [11][12].

A useful place where the data is placed is a relational database that can be defined as a server where the data is stored where the information are organized without modification of other. The data can be organized and retrieved more easily. This data can be searched with a database query. Queries are sent to the Database server using the SSMS (SQL Server Management Studio) and the data are retrieved and stored in tables. This procedure is also done by C# using the SQL connection class namespace, that permits to exchange data. The class establishes a connection between the Database and the programming language like a server-client method. The queries are sent by SqlCommnad that takes care of the connection to the database and the execution of the query. The information retrieved are stored in list, array or variables. The language used to build query is called Data Manipulation Language, and two widely used examples are the Select used to retrieve row from the database and the Insert used to insert rows in the server [13][14].

A Graphical User Interface (GUI) is used to interact with a user. The GUI framework, a combination between XAML, C#, VB.NET and others, used is called WPF or Windows Presentation Foundation. It is present inside the .NET Framework in the System.Windows namespace. The core is a rendering engine very powerful and adaptive with the modern hardware. This allows inserting in the application elements like textbox, list-box and buttons. It handles all the possible user interactions between the user and the machine allowing the developers to build in the code the necessary function for the interaction. WPF is built without the standard Windows control, so every component can be programmatically personalized. To support the communication with the user, it uses routed agent and mouse positioning. WPF can bind data when is loaded in the application managed object taking the data from inside the control, in the case the data are modified,

it takes care to load the modification also in the original managed object [15][16].

When the code is developed, it is ready for the compilation process. More precisely, the code becomes a managed execution process. The CLR prepares the runtime feature used to compile the code. The C# source code is translated into Microsoft Intermediate language (MSIL) that can be converted into native language regardless of the type of Central Processing Unit that the machine uses. The intermediate language includes all the sets of instructions for initializing methods and object for example. The MSIL is converted in CPU-specific language. This task is accomplished by the Just In Time Compiler that translates the code in different CPU architectures in fact, it is possible to target the MSIL for different CPUs so the code can run in different types of computer. The language also produces metadata that describes the types in the code and code references and the runtime used during the execution time. The metadata and the code are verified. The CLR prepares the infrastructures necessary to manage the execution of the program and the application is executed [17].

2.3 Communication with the GSM network

The modem is an acronym of Modulator-Demodulator. It allows the computer to connect to GSM network or internet. Its main task is to convert digital signal to analog signal. The GSM modem needs a connection to the GSM network for operate. The Global System for Mobile Communication is based four elements:

- Mobile device: for example, mobile phone or another device such as tablet.
- Base station sub-system: network devices that permits the communication between mobile devices. It involves the mobiles devices that need to communicate and the antennas (Base Transceiver Station) that make a bridge with them. To establish better communication the antennas are grouped in Base Station Controller, sharing radio resource and control items.
- Network Switching Subsystem: control and interface for the whole network. It is formed by switching node where the calls are addressed to the correct antenna. It has also a database which are stored the information of the subscriber. A SMS gateway handle the messages from the source to the destination.
- The operation support subsystem checks the load of the traffics inside the network.

GSM is a voice network with slow data transfer, to increase speed during the data transfer, the GPRS (i.e. General Packet Radio System) was introduced. To make the mobile device more reliable, the GPRS antenna was combined with the GSM ones. They have the characteristic that they can operate alongside the same network using the same Base Station Transmitter [18][19][20].

3 REQUIREMENTS ANALYSIS

The Automated Inventory prepares a file with all the data collected. An automated email is sent to the Selling Manager of the commissioning company. During the process, a pre-made Excel file is filled with the stocks number of each products and other information regarding general working supply are inserted as well. For example paper used to prepare gifts, lights and similar.

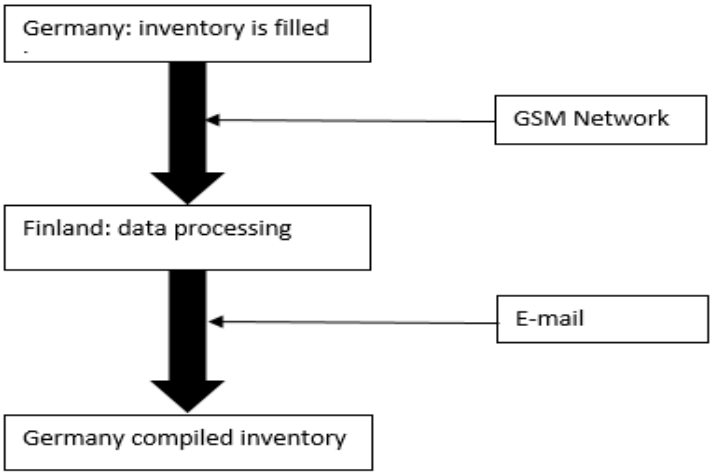
To reduce the number of human errors in writing and copying numbers from one place to another, a simple Android application with product name and input filed is developed.

The Automated Inventory application needs a stable connection to the GSM network. The network is used in the traditional way to send and receive SMS because it is one of the inexpensive way to do it. The SMS technology is old, but it is also solid and realiable. With the modern Android phones, it is difficult to retreive the messages, because they are sent in the phone database. For the purpose of this thesis, the best option is to use an external GSM modem, because retrieving an SMS is simplier than making a query every time to retrieve the message from the phone memory.

This old system of communication between hardware and phone network, is used to send and receive information within the modem. It is also used to prepare the modem communication. It sets the hardware in data mode, and other two important functions are receiving and sending command that are a key factor for this project.

The project is divided in two areas: in the client side, an Android Application is developed to make easy the operation to insert number inside the message, instead to fill manually a message. The application is divide in multiple screen, each one is studied to collect data with in the same affinity. A friendly user interface make the usage of the application more easy, and the control system help to avoid errors.

Secondary, in the computer side,it is a software that has the capability to collect data directly from the message, make appropriate adjustments. The Application has also functions that check the correctness of the message, if the messages are correct, a log file for every market with product and quantity is created and the data is stored in the database. As next phase, the automated inventory application fill in the already made Lappi-Hunaja's inventory Excel file. When all the file file is filled, the file is attached to an automated email and it is sent to the management of the company.



Picture 2. The workflow of the system.

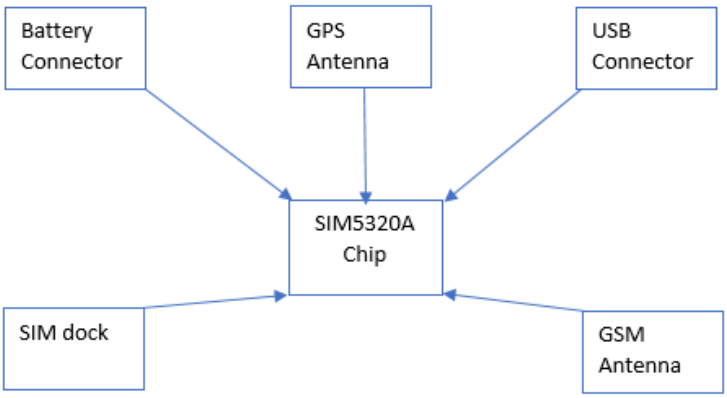
3.1 Implementation

To remain connect to the network, the application needs external hardware. In this application, is connected to a Adafruit FONA 3G Cellular + GPS Breakout and it has mounted one of the most powerful GSM module that performs all the required functions from messaging services to sending email. The hardware needed include:

- GSM processor, the SIM5320A,
- the battery connector, important to power on the module,
- the micro-USB connector, that has two principals functions:
    - the ability to exchange data between the modem and the computer, and
    - the ability to charge the battery when power is weak.
- In the bottom is present SIM dock, the place where the SIM will be inserted.

Important components: [21]:

- it measures about 4.445 centimetres of length and 4,04 centimetres of height.
- A Quad-band antenna that could be connected to any global GSM network
- Dual-band UMTS/HSDPA 900/2100MHz WCDMA + HSDPA
- GPS Antenna
- Send and receive GPRS data
- AT command interface
- USB support



Picture 3. Adafruit FONA 3G Cellular module

The AT commands are also useful to determine information about the modem to build a solid modem section with information. The automated inventory application has to display information about the modem such as the quality of the GSM quality signal, the status of the modem [22]. This command could be used directly on a terminal, or it is possible to “ask” C# to send command directly to him. This is done with the C# class “IO Port” that activate a communication with the modem that is plugged in the computer USB (Universal Serial Bus) port.

The automated inventory need a user interface to interact with the user.

Picture 4. The Adafruit FONA 3G Module – physical view

When the WPF application is opened from the computer, the final user see with a very simple user interface, a login screen appears, asking username and password to be granted on the main window of the system. Protecting data from unauthorized users is very important to prevent security breaches in data administration. In the interface it is possible to see the date, the status of the modem, if it is online and operative with the help of the system management class. A drop-down menu with the principal action that the automated inventory application can do is present. Utilities are present, for example used to send SMS to the group of sales agents, to remind them tasks, or other important communication.

3.2 Network Connection

To be able to receive message from the GSM network, the AT commands are very useful and important. AT are commands used to control a modem. AT mean “attention”.

The Automated Inventory Application uses specific command to manage the SMS from the storage of the modem Adafruit FONA 3G. The messages inbound are stored inside the memory of the device. The Application open a serial port connection between the computer and the modem, they are connected with a USB cable. The connection as the purpose to exchange data between the two device. Almost all the GPRS/GSM modems support this technology. The sphere of interest of this piece of this works are the SMS-related command:

- AT: used to call the attention of the modem
- AT+CSCS: used to set the charset
- AT+CMGL: list the SMS
- AT+ CMGD: used to delete the SMS

The application read all the message inside the device, taken them inside the Apllication and elaborated as explained in chapter five. To prevent duplicated the SMS are deleted from the memory of the modem. Other AT command are used to check if the device is registered in the network or not as follow:

- AT+CREG: used to check if the modem is connected to the GSM network.
- AT+CCID: used to read the identification number of the SIM card

3.3 Developing the application

A window with the task to check if a modem is operative and connected to the GSM network. With the help of the Database, it is possible to access to the statistic management, where all the sales data are displayed, and where it is possible to sort them, display in other way the data, create query. This is important because the management of the company always wants to be updated with the current sales situation especially when the small company that commission this thesis need to make decision what to send later to the market The system needs to be able to provide this information. Microsoft SQL server 2016 is used to storing data because the integration with the application is very simple. It is easy to use in combination with the IDE used to create the application, Microsoft Visual Studio Professional 2017.

A list box display the current information about how many SMS are inbound. A progress bar show how many SMS are missing to complete the daily inventory. An observable connection is present, in the form of a list that shows which sales agent has already sent their message. The SMS statistics also offers the option of visualing the sales situation by displaying the sales situation by point of sales.

Everytime an SMS arrives the data is sent to the database. The data needs also to be sent to an Excel file, formed by name of point of sale and every product code. This was done previously manually, need to be filled in order to have a general point of view of the selling summarized in one page. In the Automated Inventory, it is possible to create with the “COM” class of C#. A pre-made function like “Save As” helps considerably in handling the file for example. The library allow the Excel workbook to be filled to in automatically. When the Excel file it is ready, the Automated Inventory sends the file created as an email attachment and the email to the final reader. [23]

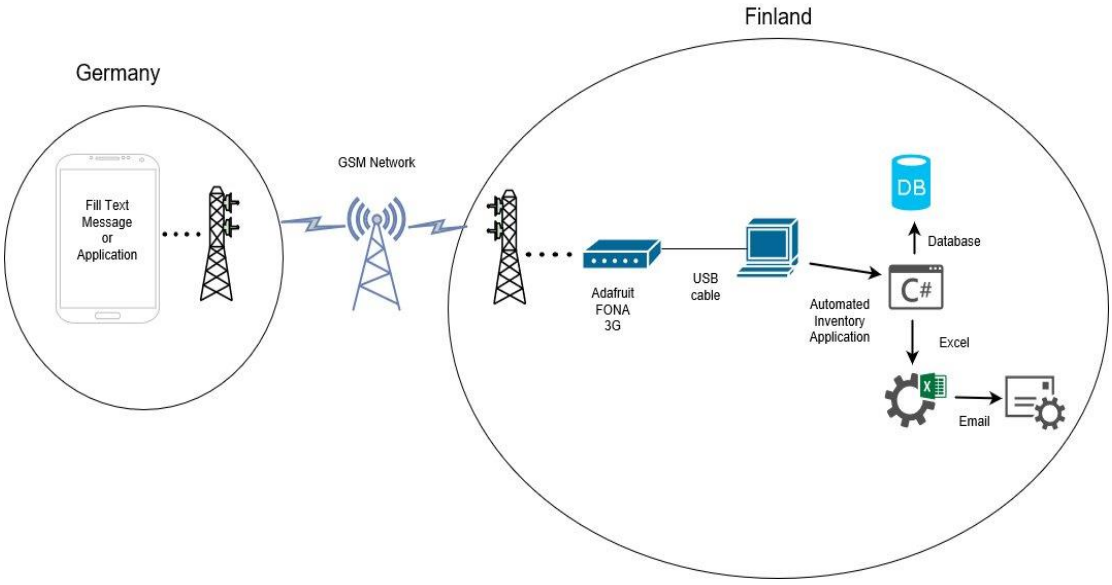
In some cases, the sales agents, has a old fashioned phone, without the Android operating system installed, the Automated Inventory offer the possibility to receive also the old type of inventory.

The new way studied in this thesis, is to use a very simple android application, also in this case with code and number, that automatically prepares the SMS and send it directly to the GSM modem. Visual Studio offer a cross platform to use C# as language to

develop the simple application. It will be composed by variuos windows, the first one with the login procedure, the user could insert its username and password inside the screen and be granted to insert the inventory. From the second window will be composed by a label and a textbox where it will be possible insert the number of the goods sold.

The difference between sending the traditional SMS and an Android application is that the second one provides a way to control the number, for example, a message appear if by mistake the sales agent wrote a negative number, and the preparation on the message will be the same every time. In the traditional messaging system, the message is already generated by the company, and the agent in the market should navigate with the cursor after the code and delete the old number and write the new one, this is very dangerous and could generate mistakes.

A picture (Picture 5) that explains how the whole system work is proposed as follows:



Picture 5. Functionality of the Application.

In the following two chapters, a full description of the Automated Inventory will be given, in both sides: client and server. These chapters are focused on the description of the application. In addition the following chapters will describe the functiononalities that allow the application to do its operations.

4 AUTOMATED INVENTORY CLIENT SIDE

The application is designed for simple usage. The visual layout defines the compound of the user interface. The language used to create it is called XML, Extensive Markup Language, which is a simple text format designed to deal with the graphical user interface of software and applications. The language used to write the engine of the application is Microsoft C#. [24]

This application is ready to be used by most Android devices, for example, also if the mobile screen is small. In this scenario, a horizontal bar helps the user to centre the screen to get the best possible user experience.

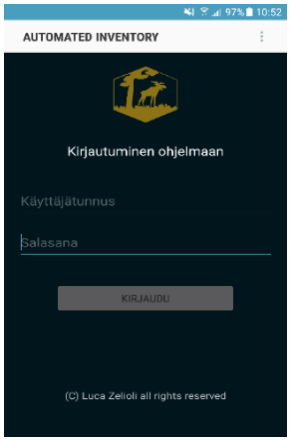
The purpose of the Automated Inventory Application client side is to collect data. The data is collected screen by screen with the application. In case the sales agents, prefer to use the traditional method of the company, they can still use it. The server side of the application could recognise these types of messages.

However, the sales agents need to start their inventory message SMS with the code “SMS” and the computer could elaborate it properly. The sales agents need to follow the structure used for the SMS structures very carefully, because in this case the error control provided by the application is not be implemented and this might cause problems on the computer side.

This is the reason why the author of this thesis has created this Android application so that the commissioner in the future could change the inventory process to a completely controlled procedure.

4.1 Inserting sales agent information

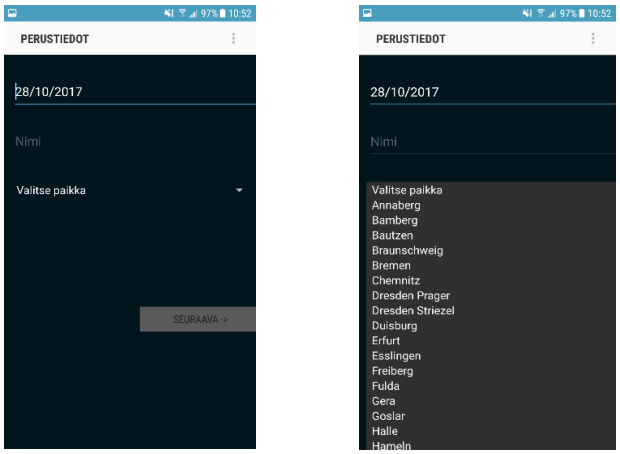
When the application is launched, a log-in screen will appear on the screen asking for a username and password. The log-in information is unique for all users.



Picture 6. Log in screen.

When the credentials are inserted, and the “Kirjaudu” button is pressed, a function to check username and password is triggered. The function (presented in Appendix 1) takes two parameters as arguments, the username and password. These two fields are linked in the user interface, and the values in there are stored in two variables. In the first step, the program is checking if one or both fields are null or empty, in which case the function stop to work presents on the monitor of the phone a message asking the user to fill in all the fields. In the second step, the username and the password are verified. In case they correspond to the ones stored in the code, a new activity is started. In android programming, “new activity” means that the program continues to the next work flow. In case that the password or username is wrong, the next activity is not triggered and a message shows it.

When the log in is granted, antother screen appears:



Picture 7. Information screen.

In the screen of the application called “Perustiedot”, (Picture 7) appears a request to fill in all the basic information of the seller agent: the date, name and a place of selling. To make the application as simple as possible, the field date is automated,i.e., it fills itself with the current date. The function takes as parameter the field where the date will be printed in, and, the instance of the date is created with “DateTime” structure. The date is stored with the format dd/mm/yyyy.

To prevent errors in digitalizing the place name, a dropdown menu, Spinner in Android programming, with all the market town name is created at this point. The list of the places is stored in the application string folder in XML format. The list of items is in the form of Array. To create the list in the user interface, there is a function that takes the array and adapts it to the layout. When a place is chosen, another function called “Selection” is triggered. It stores the place name in a variable, and checks if a selection is done, because if a user forgets to select the place, a message is triggered asking the user to make a selection.

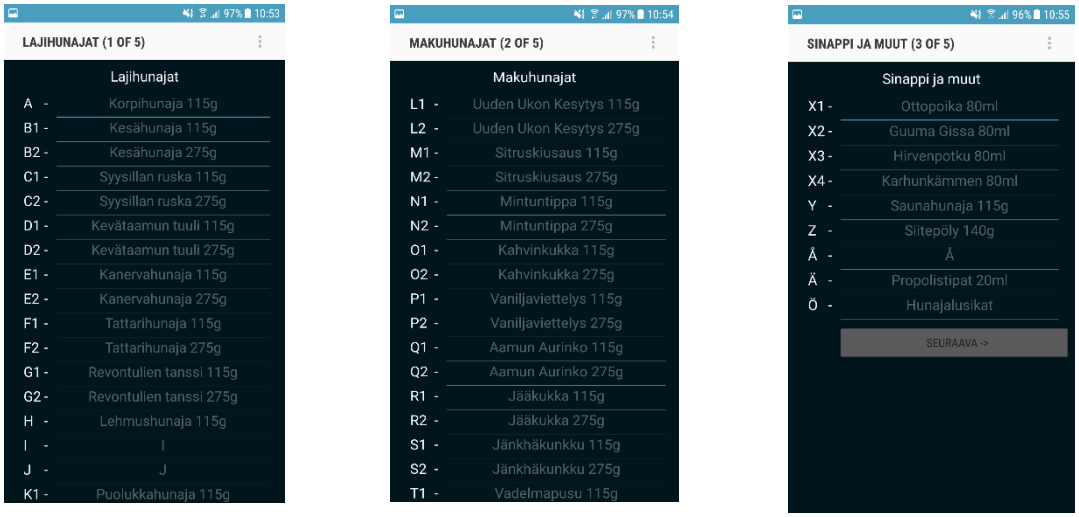
When the “Seuraava, in English next” button is pressed, *the fill the list* function is called. After the verification of the data is inserted, if some fields are null or wrong, the software places the data inside a temporary list. At this point, a class called MessageBuilder, with its method *CreateTheSMSHead* that takes the temporary list as argument, creates a string object and returns it as header of the future inventory message. A StringBuilder is the perfect class to write a long string. Its advantage is that every time a string object is created, it is immutable and every modification is reserved in memory. This class is used to modify a string without creating a new object [25].

The first “word” appended, is “APP” because when the message arrives to the computer part of this application, it needs to know where the message is coming from and take the correct branch. At this point, all the values taken from the UI are stored in the string. An instance of the new activity is created and the data is stored in the intent and passed in the next one. One advantage of this technique, the StringBuilder, is that every time a user presses the back button, a new string is created and the old one is overwritten. This work avoids completely the problem of double values which is presented when the static data type is used.

4.2 Inventory pages

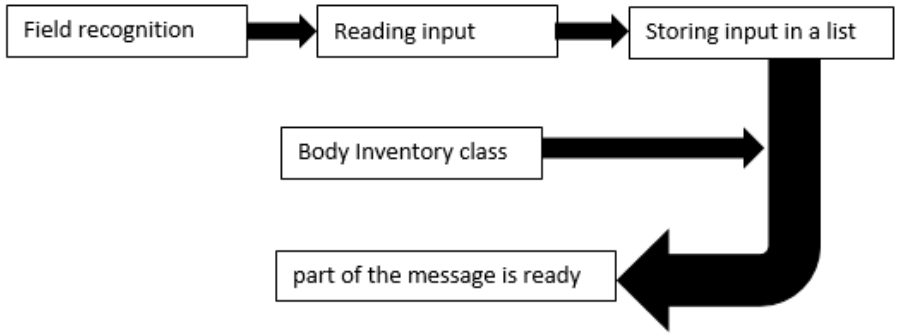
In the next three screens of the Automated Inventory application, the inventory is collected. The products are categorised as follows:

- Lajihunajat - natural honey
- Makuhunajat – flavoured honey
- Sinappi ja muut – other honey products



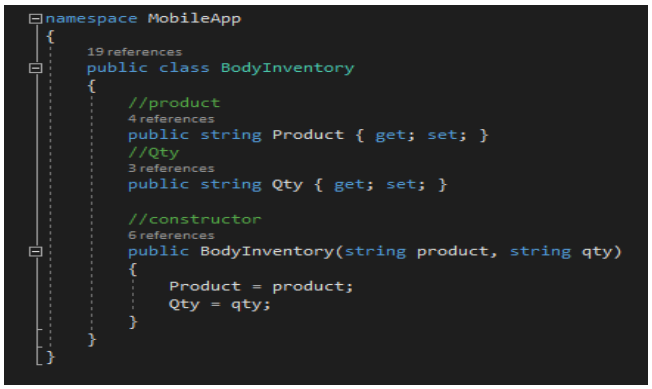
Picture 8. Different type of products.

These three parts of the application are done in the same way but the only difference is that the number of producs differs from page to page and the code of the product is different. The functions are the same for each place as described in the following diagram:



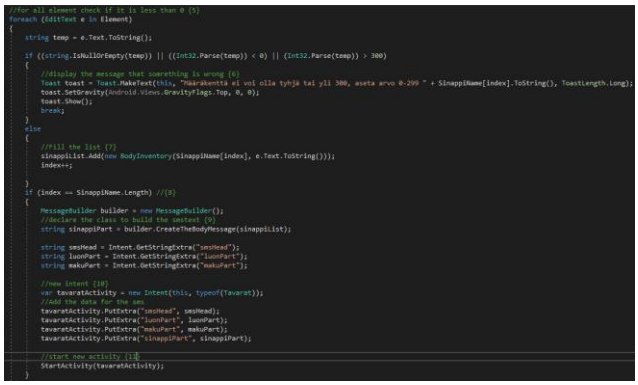
Picture 9. Description of the Inventory process.

In the beginning of the function (Appendix 4) a string array with the “code of the product” is declared, and all the fields are istantiated. To combine the code of the product with its number, a personalized list of object is declared at this point, and it is one of the keys of the whole inventory process.



Picture 10. Creation of a personalized list of object.

The object is called BodyInventory. It is formed by two strings: the products and the quantity. Instead of writing the product name that is too long, a code is used to represent it. The convenience is that the string array declared in the beginning of the function, and the quantity is read from the list inside each page of the application.



Picture 11. The loop.

Used for each loop, the code puts inside a temporary variable the content of each elements and tests it in three ways:

- the first test checks if the field is null or empty because the fields can not be empty
- the second test checks if the number is less than a zero and because every field can not contain a number less than 0.

- the third test checks if the number is greater than three hundred because the maximum amount of each product that seller could dispose is 299.

If one of these conditions are not satisfied, the function does not send the data as part of the inventory. Instead, a message will appear and display the position where there is an error.

Vice-versa, the element and its code are added to the list, and a counter variable, with the scope to count the elements of the list is incremented.

When the number has the same value of the string array length, that represents the total number of elements, the inventory is ready and the function builds the appropriate part of the message.

With a loop, the MessageBuilder class creates the part of the message relative to the screen where the final user is interacting. A new string is appended, with the name of the page as identifier, to the Android intent and in this point the next activity starts.

The last two screens of the application are particular and, they are part of the inventory but they collect a different type of data.

The “Tavarat” (Picture 12), screen is formed by a list of double check boxes where the seller must click if an object is needed or not. Also, here there is a control system and, it is not possible to click both check boxes on the same line. If one is clicked, the second one becomes automatically de-selected.

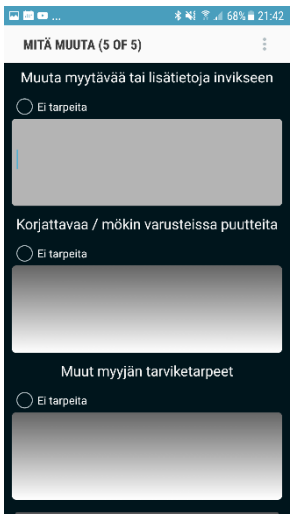
Picture 12. Tavarat page.

The software at this point goes to search the two check boxes that are present on each line and they are added again in the temporary list of this type of objects. Every time one of the two elements of each lines is selected, the other element on the same line is checked, too. If it returns a positive response, it deselects it. This procedure is done every time an object is selected, thanks to *delegate*, a special method that handles events, that attaches an event handler to the user interface. The information is sent in the same way as the BodyInventory list and the data are saved in a string object with intent. If an element of the list is not clicked, a message shows the place where the checkbox is missing and the application does not go in the next screen.

When the collection is filled with all the values from the screen, mathematics is used to check if the checkbox checked was “Kyllä” (yes) or “Ei” (no). A loop goes inside the list and if the remainder of the division between the counter value (the variables described above) is equal of zero, it means that it was in the first position of the screen, Kyllä, or if it was one, it was in the second position of the line, with a value of “Ei”. When the counter reaches the number of elements, a part of the message is built, and user is allowed to click “next” and pass to the next screen.

The purpose of the last screen is to allow the vendor to write if supplementary goods are needed in the place. It is divided in three multiline boxes, where it is possible to write, for example, what else is missing and if something needs to be fixed.

To make a specific control and avoid mistakes or avoid oversights, a checkbox must be checked if the final user does not want to write comments.



Picture 13. Last inventory screen.

The checkbox can disable the multiline-box if it is pressed. The loop, presented in Picture 14, helps the application to control the flow of the work in this part of the code. The content of each multi-line text box is transferred in one temporary variable. The program checks first if the check box is checked, so no words are inside the box, the function looks into the string variable and, if it is empty the word “Ei” is appended in this part of the inventory. Vice versa, if the box is filled with comments, and the check-box is checked, a message comes with the position of the mistake. If the checkbox is un-checked, the program checks if the string is not empty, in which case it does not return any error and the text is saved with the position in the inventory. When the counter variable, is called, in this part of the program, the part of the message is built and the whole message is ready to be sent.

```
foreach(CheckBox c in checkBox)
{
    string temp = edit[i].Text.ToString();
    //check every checbox is clicked or not
    if(c.Checked == true)
    {
        if (!string.IsNullOrEmpty(temp))
        {
            Toast toast = Toast.MakeText(this, "Checkbox is checked and editText is full " + korijaName[i].ToString(), ToastLength.Long);
            toast.SetGravity(Android.Views.GravityFlags.Top, 0, 0);
            toast.Show();
            break;
        }
        else
        {
            product[i] = "Ei"; // add in the position not
        }
    }
    else
    {
        if(string.IsNullOrEmpty(temp))
        {
            Toast toast = Toast.MakeText(this, "Checkbox is not checked and editText is empty " + korijaName[i].ToString(), ToastLength.Long);
            toast.SetGravity(Android.Views.GravityFlags.Top, 0, 0);
            toast.Show();
            break;
        }
        else
        {
            product[i] = temp; //ad the text there
        }
    }
    i++;
```

Picture 14. Loop that manages the last screen workflow.

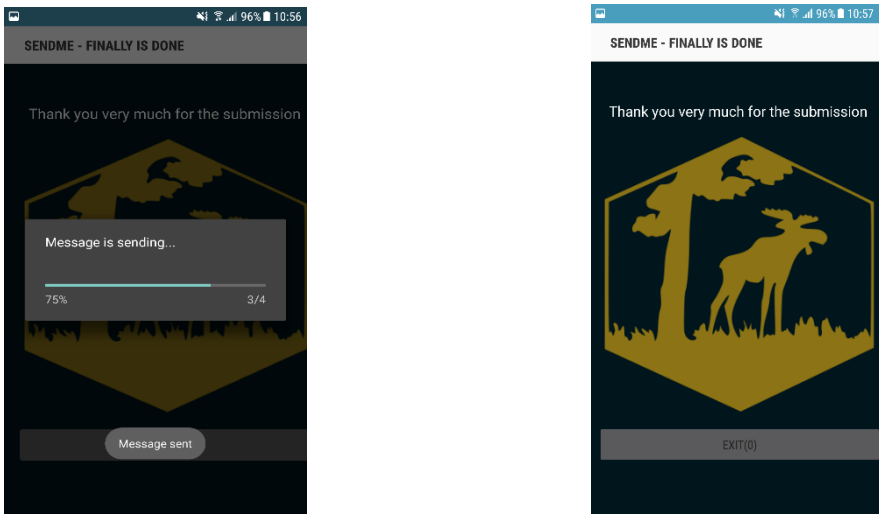
4.3 Sending the SMS

In the last part of the application, the work is done with the background of the exit screen. The SmsManager is a class that manages operation for managing SMS by calling the static method getDefault and, the parent class is the Telephony class [26].

```
string message = smsHead + " " + luonPart + " " + makuPart + " " + sinappiPart + " " + tavaratPart + " " + korjaPart;
```

Picture 15. Building all the SMS body.

Before the message is sent, the class SendMe (Picture 15) prepares all the body of the inventory attaching all the string builded in all the previous messages body create in all the screens of the application.



Picture 16. Last screen.

When the message is created, the function Send inventory that take two parameters, the message body and the phone number, is sent. The body of the function is shown in Picture 17.

```
1 reference
protected void SendInventory(string phone, string inventory)
{
    //instance of new sms
    SmsManager sms = SmsManager.Default;

    IList<String> parts = sms.DivideMessage(inventory);
    int num = parts.Count();
    numberForProgressBar = num; //add here the number for progress bar max = number sms
    List<PendingIntent> sent = new List<PendingIntent>();
    List<PendingIntent> delivered = new List<PendingIntent>();

    for(int i = 0; i < num; i++)
    {
        sent.Add(PendingIntent.GetBroadcast(this, 0, new Intent("SMS_SENT"), 0));
        delivered.Add(PendingIntent.GetBroadcast(this, 0, new Intent("SMS_DELIVERED"), 0));
    }

    //send it
    sms.SendMultipartTextMessage(phone, null, parts, sent, delivered);

}
```

Picture 17. The function used to send SMS.

A instance of the class SmsManager is called and is instantiated. The string that contains the inventory is tested to check if it fits a single message, or it is divided in multiple parts. In the meanwhile, two intent lists with the messages send and delivered are prepared.

These intent lists are important because they receive as feedback, the response, or better the status of the messages sent. The method that permits the message to be sent is called SendMultipartTextMessage and takes as argument the phone number, the parts of message, and its intents.

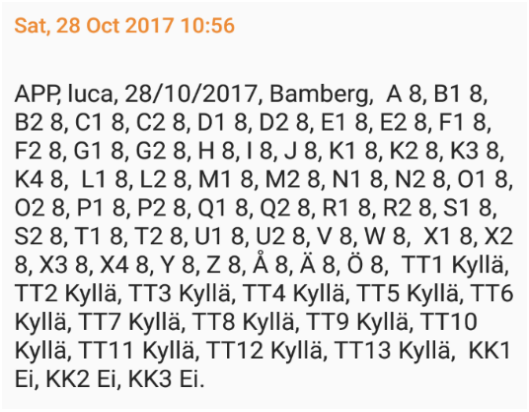
When the message is sent, a report of messages appears on the screen of the phone, as a progress bar which takes as maximum value the total number of messages, and every time, one is sent it progresses one by one (Picture 16).

The progress bar works in an effortless way. It is in the Broadcast receiver class, a special part of Android Operating System which manages the report from the phone, for example, the report when a message is sent, or the quality of the GSM signal. The function in Picture 17 sends an intent to the broadcast receiver. It contains instructions for Android to “show” the report of the message, i.e., if it is sent or it is lost. Every time a message is sent (usually are 4 or 5 per inventory) a counter makes the progress bar increase its value. The maximum value is the number of messages prepared from the application.

A further plan of the application will be the implementation of a more precise inventory system control. The management of the company could insert the amount of stock sent to each market. When an inventory is prepared, the application consults the online database to check if the number inside the inventory field is consistent. This can be done by comparing the number inserted and the amount of honey stock in the sales point. This result could be achieved by utilizing a technology provided by Microsoft Azure.

To pass data between the screen, a static list was implemented.The static list can not be deleted until the program is closed. The idea was to pass a static list in all the Android screen, and feed it every time with inventory data, but a problem was discovered. When using this type of data collection in the application, a problem appears if a user presses the back button of the phone, the list remains but it duplicates the values for the second time. Because of this problem, the structure of the application was changed. To pass data between various screens, an intent needs to be used. An intent is an abstract description of the operation that will performed as next. In this place, it is possible to start new activities, and pass data between them [27].

The Android application creates the following output (Picture 18) as SMS and sends it to the server side of the Automated Inventory.



Picture 18. The SMS sent by Android application.

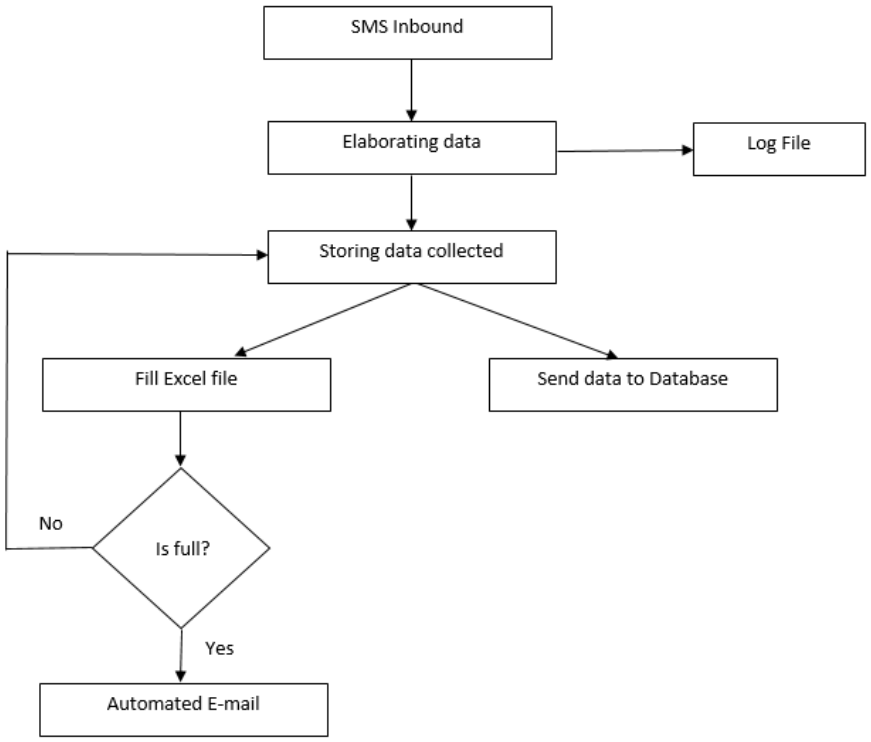
The next chapter of this thesis analyses the server side of the application following the same methodology used in this chapter. The chapter also describes the procedure used to manage the received SMS from the application.

5 AUTOMATED INVENTORY SERVER SIDE

The Automated Inventory Application server side has the main task of collecting inventory data from the various sales agents. This is accomplished in six steps:

1. The Automated Inventory Application search in the modem the inbound SMS
2. The messages are moved in the memory of the application
3. The content of the messages is moved in a list of products id and quantity
4. The content of the list is placed in the company Excel file which contains the summary of the selling data for every market.
5. The same content is also moved to the Database for further sales statistics.
6. When the Excel file is full filled, an automated email is sent to the management of the company.

The following diagram summarizes the process of the application:



Picture 19. The Workflow of the application.

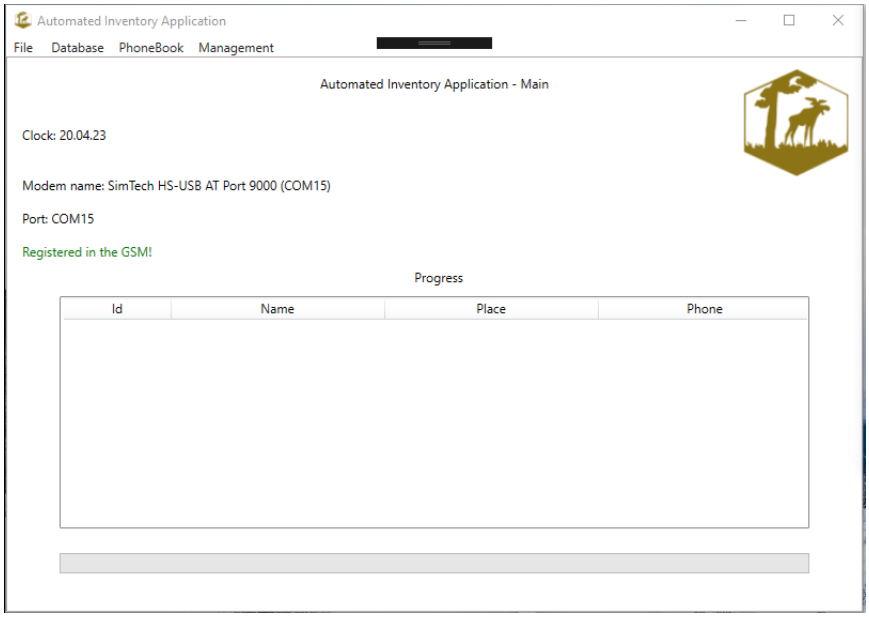
The software package is divided in five folders. Every folder has its own competences:

1. Database is a folder that contains the class that manages all database's operations.
2. Excel is the folder that contains the class that manages Excel operations.
3. File is the folder that contains two classes: the first one called EmailServices is used to manage the automated email, and the second FileManagement has a task to create directories, subdirectories and it is also responsible of creation of a log file for all the software.
4. Management contains three classes, the first one is responsible for the management of the applications: modem recognition, port of the modem, GSM network status, modem battery status. The second and the third one are mainly tasked to manage SMS.
5. The Var folder contains the base of the company´s Excel file used to fill the inventory.

In the following sections, an analysis about every step that this software uses to accomplish a fast and reliable inventory is described.

5.1 Graphical user interface

When the application is launched, a window prompting username and password appears on the screen. The graphical interface is very simple: two textboxes, one for the username and the second one for the password. To guarantee security, the username and password are encrypted with the SHA algorithm. The SHA cryptographic system, used by the American National Security Agency until 1995, is a cryptographic system that protect data from unauthorized viewer. When the button Log-In is pressed, the username and password are stored in two variables. The Application, call the Database Management class passing the two variables in the Login function. At this point, the class makes a connection to the database, it retrieves the encrypted username and password it decrypts it with a function called hash and it checks if they correspond with the ones stored in the variables from the windows login. If the procedure is successful, the main windows of the software open.

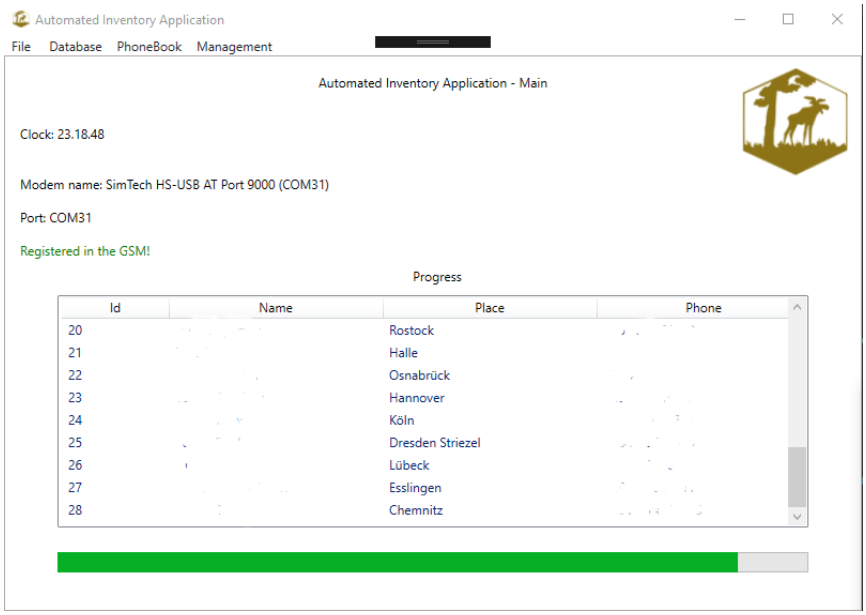


Picture 20. Main View Automated Inventory Application.

In the background, before the main windows are loaded, the software starts the operation to retrieve the modem connection information that are needed to run. The function returns a string array called the ManagementObjectSearcher and it retrieves the modem from the windows management database. The modem name is stored in the position zero of the array, the position number one stores the COM port that the software uses to communicates with the modem. This is a crucial operation for the Automated Inventory Application because without a port that communicates with the modem, the application is only a software that stands alone. In position number two, the status of the GSM connection is reported. This is achieved with the AT command “AT+CREG”, which checks if the modem is registered on the network. If one of these three operations has a negative response, an exception is thrown. An error message appears on the screen and the FileManagement writes the error in the master log file. When the operation is completed, all three pieces of information are shown in the window.

The main window has a simple user interface. It displays a clock, the USB modem name, the communication port number and the status of the GSM connection, that are filled with the value of the function described above. There is a table that collects the identification number, the name, place and phone number of the sales agents that already have sent their inventory. The table is a list box divided in four columns, where every column binds an element of an Observable Collection. This collection is very useful because the content is reported in real time every time a change happens. In this case,

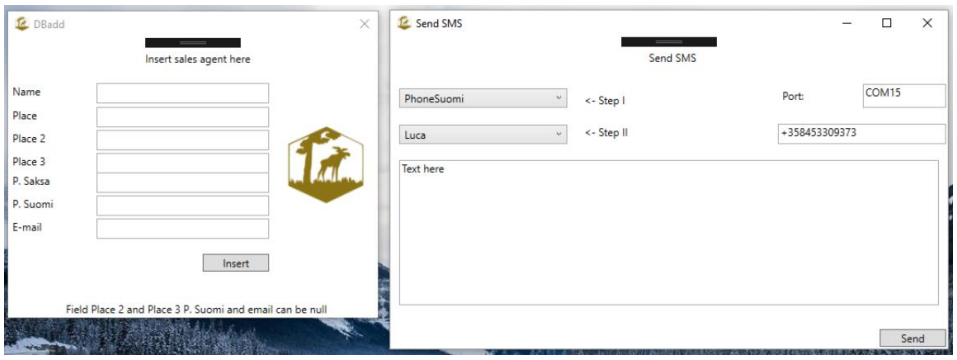
it is filled when an inventory operation is completed (Picture 21). To create an Observable Collection, a personalized list is necessary. In the case of Automated Inventory Application, the list is taken from the SMS header when it arrives in the computer memory. The personalized list is called “Observable” and takes as elements the inventory progress number, the identity of the sales agent, the phone number and the place. This is an effective way to have a “live” point of view of the progress of the inventory. An agile window like the main one of Automated Inventory Application, could give a frame of the situation, without pressing another button or navigating in a different window. A progress bar shows the number of inventory that are already inbound and how many are still missing. Every time an inventory is completed, the bar adds one value to the total progress of the inventory, when the progress takes the maximum, the system sends an automated email to the management of the company, with the summary of the inventory as an Excel file attachment.



Picture 21. Observable Collection with filled data. All information are removed.

A dropdown menu, present in this application, makes a simple navigation between the various windows of the program. It is formed by a dock panel that contains the category of menu and every category contains its own submenu. The menu file contains the submenu exit, to avoid closing the window by mistake, a confirmation message appears. The Database menu is added in order to retrieve from the inventory an extensive pool of statistic related to the sales trends. However, this implementation is not ready yet. The phone book menu contains two submenus (Picture 22). The first one is used to add sales

agents to the phone book, the second one is used to send the messages to the sales agents. It contains a drop menu that selects which phone book to use, the German number or the Finnish number. The second list contains the name of the sales agent, and when selected, the number is added and the system is ready to send the message.



Picture 22. A view of the phone boook and SMS sender system.

The SMS is sent with the function SendSMS, in the Send class that takes two parameters, the phone number and the body of the SMS that are collected from the window. The software opens the COM port to communicate with the modem, it sets the modem in message mode with the AT Command “AT+CMGF=1”, it sets the number of the receiver with the “AT+CMGS”, it appends the text and the message is sent. In the end, it closes the COM port. A message with the status of the SMS sent appears when the window is closed. It is possible to see the function in the Appendix 2.

In the Modem information window (Picture 24), it is possible to select a suitable modem to use with the software. It is possible to select the device from a dropdown menu. When this menu is clicked, the Automated Inventory Application calls a function that feeds a list of USB modem attached to the computer and inserts it in the menu. The modem name is selected from the Windows Database of the computer and the software asks for the caption where it is possible to find the modem name and the COM port used.

The port is given by the Regex technique that handles regular expressions, and it is used to find some specific character or expression in a string (Picture 23).

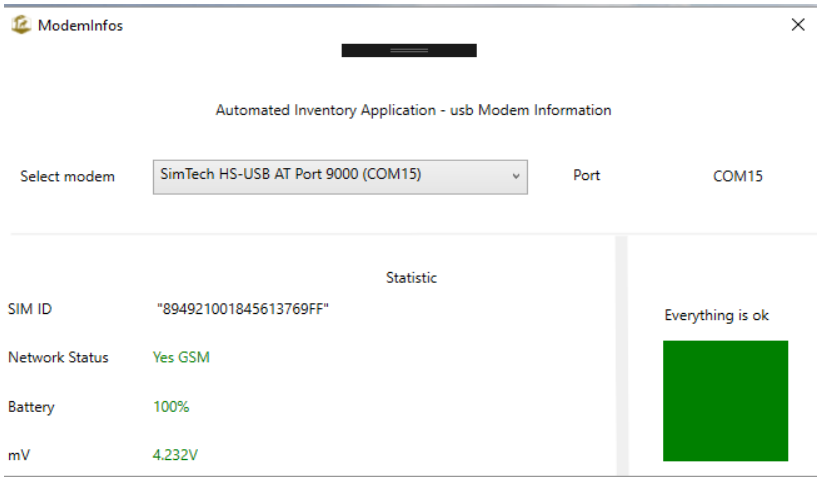
```
//com Port[1]
modemInfo[1] = Regex.Match(modemInfo[0].ToString(), @"\(([^)]*)\)").Groups[1].Value;
```

Picture 23. The COM port is achieved with Regex technique.

The other statistics, that are shown in the window, are acquired within the usage of AT Commands. For example, the SIM Identification number is retrieved with the command “AT+CCID” and it returns a very long string with plenty of useless information. In this case the technique used to retrieve only the identification sim number is “String.Replace”. This command replaces patterns of a string with empty string it replaces the substring with empty space. The registration status in the network is achieved in the same method of the background operation which the main windows of the application uses to load this information inside the main page.

The FONA 3G module needs an external battery to run, so in the modem info page there is the battery status in the statistics. The battery status is retrieved with the AT Commands “AT+CBC” that returns the battery status in percentage and its voltage. If all the information is retrieved correctly, a green square appears in the Modem Information window. In this window, there is a warning system and the square changes colour as follows:

- The square changes colour in yellow if the battery is under 50%.
- The square changes colour in red if the battery is under 20%.
- The square changes colour in black if it is present a malfunction for example if the software can not recognise the USB modem.

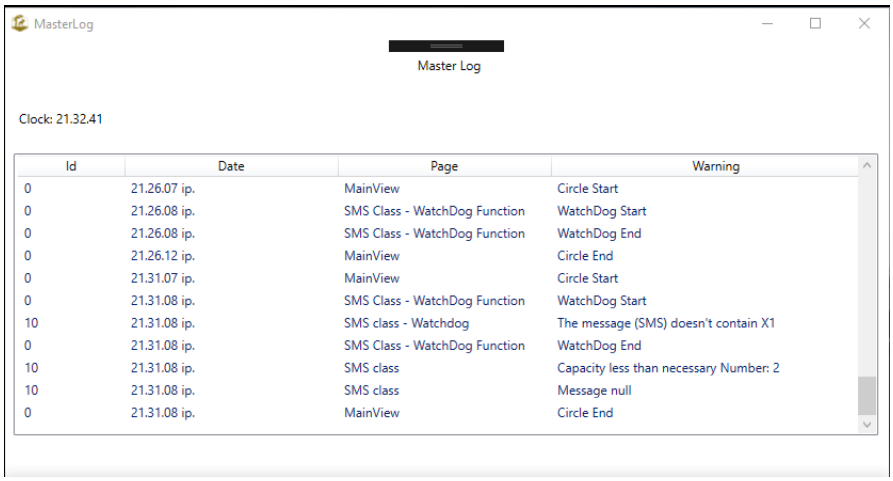


Picture 24. Modem information window.

Another important window is the Log-report, called Master Log (Picture 25). It is composed by an Observable Collection that reports all the events registered in the software. Every time an event is triggered, the event is reported and shown in the list. The list is an Observable Collection which shows data in real time when an inventory

process is called or when a mistake is found inside the SMS. It also presents the severity of warnings that indicate if the messages prompted is just a warning, or if the software needs deeper attention.

The Severity goes from zero to ten where zero is acknowledgement of an event (for example, the end of inventory circle, described in next chapter) and ten is a dangerous situation which means that an error or an exception in the software has occured. The log window also indicates where the problem or the warning is found, the page of the code file and the function name, so that it is faster to solve a problem situation, and it is possible to examine directly the problem. All the warnings are also reported in the log file that is created every time the Automated Inventory Application is started.

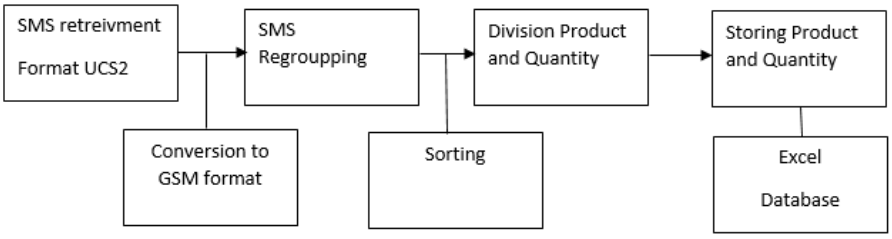


Picture 25. The Master Log window.

The next sections offer a deeper insight of Automated Inventory processes. They also describe how the classes involved in this process work and analyse the process from the retirement of the Unicode message and its arrival to the database and to the company Excel inventory file. An analysis is also made, in the next chapter, of the sophisticated log-file system of the Automated Inventory Application.

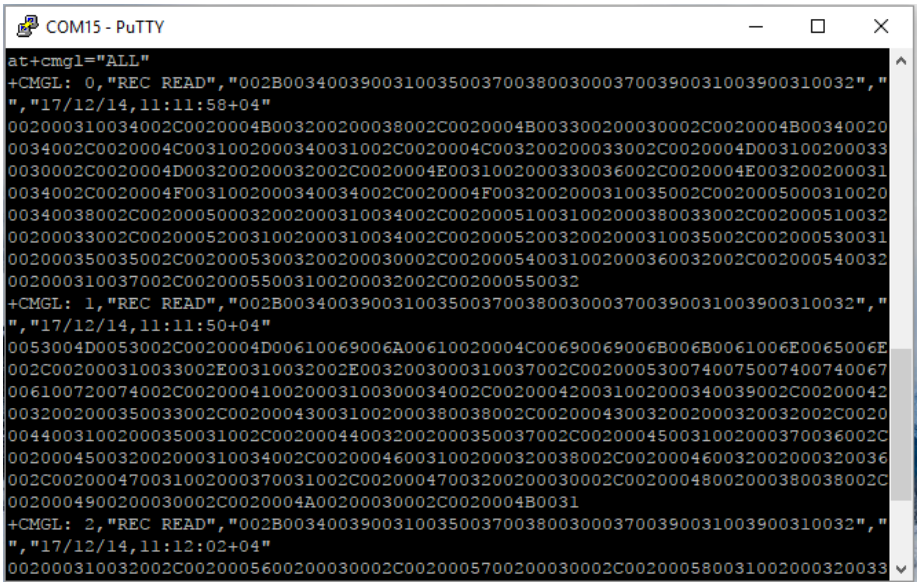
5.2 The Inventory process

The Inventory process could be summarized in the following diagram:



Picture 26. Diagram of inventory process.

All the functions necessary to do the inventory are enclosed in the SMS class of the Automated Inventory Application. When the application is launched, in the background of the application a timer of the duration of 5 minutes is triggered. When the time is elapsed, a function called “do_Inventory”, inside the initiator, is triggered. The application sends a request to the FONA 3G asking to retrieve all the SMS inbound, transporting it in the computer memory. The message is received by the modem in the UCS2 format. It is a format where the message is represented in Hexadecimal notation. The content is difficult for human to read (Picture 27).



Picture 27. Example of multiples SMSs in FONA3G memory.

The commands used to achieve this operation are:

- AT+CMGF=1 this command is used to set the GSM modem to message mode.
- AT+CSCS=”UCS2” this command is used to set the character of the message to Unicode.
- AT+CMGL=”ALL” this command is used to retrieve all the SMS reads one or not.

The next operation that the software does is to convert the messages, in GSM format, human readable, and takes only the ones with the same phone number and leaves others for further rounds. If the content of the message is empty or the number of characters is less than 97, the message becomes null and the cycle starts again. Ninety-eight is the minimal number of characters that a message should contain. This number comes from a study when all the messages of past years were analysed with the result that the shortest SMS had 98 characters. If the SMS are retrieved successfully, they are stored in a variable.

This variable, which data type is string, is passed in a function called GetTheMessage, and a new “Regex Class” is called:

```
Regex regAll = new Regex(@"\+CMGL: (\d+),""(.+)"",""(.+)"",(.*),""(.+)""\r\n(.+)\r\n");
Match matchAll = regAll.Match(message);
```

Picture 28. Regex class declaration.

The Regex ignores everything before the AT Command response, “+CMGL” which has the significance that SMS are found inside the modem memory, and divides the patterns in 6 groups:

- Group one is the progressive number of the inbounds messages
- Group two contains extra information of the message, but is not used in this application
- Group three contains the phone number of the SMS
- Group four contains set of characters not used for this application
- Group five contains the data and time of the message
- Group six contains the body of the message

Multiple patterns are found at this point, so the method “Match” stores the set of patterns inside the match object. At this point, the Automated Inventory Application takes the body of the message and the phone number, and stores it in variables. The program searches inside the body of the message for the key words “APP” or “SMS”, if both are not present, the application searches for another pattern until the key words are found. As mentioned in the previous chapter, the Automated Inventory Application server side has the capacity to manage messages received from the application as well as from the normal SMS messages sent from a phone. When the pattern match, “APP” or “SMS” is found, the phone number is saved and loop is interrupted (Picture 29).

```
while (matchAll.Success)
{

     decision = ConvertUCS2toReadableFormat(matchAll.Groups[6].Value);

    if (!decision.Contains("APP") && !decision.Contains("SMS"))
    {
        matchAll = matchAll.NextMatch();
    }
    if(decision.Contains("APP"))
    {
        decision = "APP";
        newPhone = ConvertUCS2toReadableFormat(matchAll.Groups[3].Value);
        break;
    }
    if (decision.Contains("SMS"))
    {
        decision = "SMS";
        newPhone = ConvertUCS2toReadableFormat(matchAll.Groups[3].Value);
        break;
    }
}
```

Picture 29. Branching between APP and SMS.

The application searches for the beginning of the string the keywords “SMS” or “APP” and stores it in a variable called decision that represents the type of the message received. Thanks to this variable, the software, takes the correct branch, i.e., correct way, because it needs to sort the position of the body of the messages. This operation is important because when the modem receives the messages, the arrival does not match the sending order, but they always arrive in different one. Both body messages from the application and from the normal SMS, are sorted based on key words (Appendix 2 f). For the application case, the key words were found sending multiples times inventory from the Android application, with different type of numbers, product quantity with one number, with two number, with three number, and with chaotic sequences with the following results:

- The pattern that contains “APP” is always in position 0, the head of the message.
- The pattern that contains “P1” is always in position 1, it stays right after the APP pattern.
- The next patterns always contain “Ä” (position 2).
- The next patterns always contain “TT12” (position 3).
- The other messages are appended after these.

For the normal SMS case, the division is carried out by studying the past years messages, and the statistic are made as follows:

- The patterns that contain “SMS” are always in position 0.
- The patterns that contain “P1” are always the next one (position 1).
- The patterns that contain “Ö” are always the next one (position 2).

- The other messages are appended after these.

Every time a pattern is analysed, the phone number needs to match the one stored in the beginning of the procedure, where the software searches inside the message for the key words “APP” or “SMS”, if the number does not match, the sample is discarded. When the position is set, the body of the message is inserted in the following list:

```
18 references
public class ListBuilder
{
    2 references
    public string Id { get; set; }
    2 references
    public string Number { get; set; }
    5 references
    public string Body { get; set; }
    1 reference
    public string Time { get; set; }
    3 references
    public int MessageId { get; set; }

    //constructor
    2 references
    public ListBuilder(string id, string number, string body, string time, int smsId)
    {
        Id = id;
        Number = number;
        Body = body;
        Time = time;
        MessageId = smsId;
    }

}
```

Picture 30. List that collect the multiple messages.

Where Id, i.e. identification, is the position in memory of the message received inside the modem, number is the position assigned to the message, the body of the message, the time that the modem received the message and message id where it is possible to see the order of retirement without sort. At this point, the list is sorted using comparison sample the string Number using “list.Sort” method. When the list is complete, the data inside is passed in a Delete function (Picture 31) that takes the list as parameters:

```
1 reference
private void DeleteTheSMS(List<ListBuilder> list)
{
    sp.Open();
    //attention to the modem
    sp.WriteLine("AT" + Environment.NewLine);
    Thread.Sleep(300);

    foreach (var s in list)
    {
        sp.WriteLine("AT+CMGD=" + s.Id.ToString() + Environment.NewLine);
        Thread.Sleep(300);
    }

    var message = sp.ReadExisting();

    if (message.Contains("ERROR"))
    {
        MessageBox.Show("Error in delecting the message from the modem", "Automated Inventory Application", MessageBoxButton.OK, MessageBoxImage.Error)
        mgt.Warning(10, "SMS class - Delete Function", "Error in delecting the message");
    }

    sp.Close();
}
```

Picture 31. The Delete function.

The task of the function is to delete the messages elaborated until this point, because it is not necessary that they stay in the memory of the FONA 3G modem. The command used in this case is called AT+CMGD =" Identification of the message" where identification of the message is the integer store inside the list shown in picture number (Picture 31).

The message is now passed as argument of the SmsProdAndQty function. The message is built as a long string with the usage of the string builder. The body of the inventory is split with the "list.Split" method (Picture 32), to get rid of the spaces and comma, used as separator. The automated Inventory Application takes the name of the sales agents, the phone number, the type of message, the date and stores it as header element of the list SmsTemporanyStorage.

```
string[] temp = builder.ToString().Split(new char[] { ',', ' ' }, StringSplitOptions.RemoveEmptyEntries);

//get base info
phone = temp[0].ToString();
messageType = temp[1].ToString();
```

Picture 32. Getting Product and Quantity as data.

The method "list.Split" can divide the string according to certain characters, in the case of this Application, the characters are represented by the "*new char [] {',' ' '}*" a comma and a space. Every time the method finds this set of characters, it creates a new element of the array. The application takes the temporary element inside the array and puts it in the SmsTemporaryStorage, dividing it in products and quantity.

When it encounters these are product codes:

- SS for the normal SMS.
- KK1, KK2, KK3 for the Android application.

A loop rebuilds all the sentences or comment and stores it as an unique element. This is done because in the original message, when the method "list.Split" arrives at this point, it finds a space and puts every word as new element of the array, but in this case it is important that this part of the inventory stays in one sentence. At this point the inventory is collected in the list and is ready for storing in the excel file and Database.

5.3 Storing Data

The inventory data is ready, in human readable format and it is stored in a list. The Automated Inventory Application prepares the folder where the Excel file with the summary of the inventory is saved. This is achieved by the FileManagement class. The class verifies that the main folder that collects all the subfolders of the files is present, and it creates inside a new subfolder with the date of the inventory as name of the folder (Appendix 2 g).

At this point, the Automated Inventory Application invokes the “ExcelManagementClass”. The function takes all the elements inside the smsTemporanyStorage and appends it to the company Excel inventory file:

```
foreach(TemparanyPlace p in place)
{
    if(list[0].Place.ToString().ToLower().Contains(p.Place.ToString()))
    {
        col = p.ID;
        break;
    }
}
try
{
    foreach (SmsTemporaryStorage sms in list)
    {
        if (sms.ProductName.Contains("TT1")) //if contain TT1 there is a gap in the file excel
        {
            row = 58;
        }
        if (sms.ProductName.Contains("TT10"))
        {
            row = 67;
        }
        if (sms.ProductName.Contains("TT11"))
        {
            row = 68;
        }
        if (sms.ProductName.Contains("TT12"))
        {
            row = 69;
        }
        if (sms.ProductName.Contains("TT13"))
        {
            row = 70;
        }

        if ((sms.ProductName.Contains("KK1"))) //if contain KK1 (SS for normal sms) there is a gap in the file exc
        {
            row = 72;
```

Picture 33. Part of loop function filling the Excel file with inventory.

In this function, there is a distinction between the normal SMS and the APP, and the address in their respective cell in Excel. For every inventory retrieved the ExcelManagement class, checks the name of the market and address in it. This is done with a comparison between the name of the market inside the SmsTemporanyStorage and an internal pre-made list in this class. The temporary array that contins the name of all markets place, is looped until the market name is found and it returns the code and the column number where the class appends the inventory number. In the following page, a series of Pictures of the Excel file are presented.

Koodi	Hunajat	Koko	Braun-schweig	Hanno-ver	Hameln	Soest	Köln	Duis-burg	Müns-ter	Osna-brück	Olden-burg	Bremen	Lübeck	Rostock
A	Korpihunaja	115 g	81	98	60	73	94	52	68	64	46	105	72	75
B1	Kesähunaja	115 g	108	87	63	49	105	116	49	58	81	73	17	64
B2		275 g	31	32	32	40	42	27	35	43	22	29	27	26
C1	Syysillan ruska	115 g	48	64	25	48	89	88	78	80	57	32	36	41
C2		275 g	18	30	17	30	32	22	24	19	21	23	28	20
D1	Kevätaamun tuuli	115 g	81	71	54	89	64	74	62	113	68	30	54	40
D2		275 g	23	28	46	46	22	32	33	40	38	24	30	29
E1	Kanervahunaja	115 g	46	121	26	34	95	60	40	37	41	65	124	57
E2		275 g	14	19	16	19	29	7	16	16	22	14	24	16
F1	Tattarihunaja	115 g	95	93	53	15	79	87	56	63	53	44	63	73
F2		275 g	13	14	13	26	44	37	17	35	17	23	4	19
G1	Revontulien tanssi	115 g	60	24	7	32	114	65	39	58	50	30	26	20
G2		275 g	0	0	0	0	0	0	0	0	0	0	0	0
H	Lehmushunaja	115 g	72	38	72	109	76	65	40	78	40	66	61	56
I			0	0	0	0	0	0	0	0	0	0	0	0
J			0	0	0	0	0	0	0	0	0	0	0	0
K1	Puolukkahunaja	115 g	8	3	5	6	11	0	7	11	14	14	6	3
K2	Lapin Taigahunaja	115 g	9	8	4	13	10	11	5	14	13	6	8	8
K3	Mustikkahunaja	115 g	3	4	5	6	12	8	5	11	15	9	7	3
K4	Hillasuonhunaja	115 g	11	4	11	9	13	8	8	9	14	3	7	1
L1	Uuden Ukon Kesytys	115 g	60	26	43	32	45	56	67	57	39	50	59	40
L2		275 g	18	15	8	8	13	15	16	9	10	1	6	2
M1	Sitruskiusaus	115 g	51	37	61	24	89	80	89	87	44	40	47	74
M2		275 g	9	7	6	9	15	21	16	8	6	10	7	0
N1	Mintuntippa	115 g	43	28	37	24	37	52	53	14	40	36	55	38

Picture 34. Excel file part 1.

Koodi	Hunajat	Koko	Braun-schweig	Hanno-ver	Hameln	Soest	Köln	Duis-burg	Müns-ter	Osna-brück	Olden-burg	Bremen	Lübeck	Rostock
M1	Sitruskiusaus	115 g	51	37	61	24	89	80	89	87	44	40	47	74
M2		275 g	9	7	6	9	15	21	16	8	6	10	7	0
N1	Mintuntippa	115 g	43	28	37	24	37	52	53	14	40	36	55	38
N2		275 g	13	0	0	14	14	16	17	0	7	8	16	8
O1	Kahvinkukka	115 g	35	77	37	45	50	62	57	78	52	50	68	51
O2		275 g	14	15	4	13	9	23	18	9	2	17	11	13
P1	Vaniljaviettelys	115 g	52	44	44	47	109	62	56	69	45	62	89	59
P2		275 g	15	8	9	10	6	22	18	6	6	10	10	14
Q1	Aamun aurinko	115 g	66	69	53	69	71	86	72	78	71	37	53	50
Q2		275 g	7	11	5	2	10	4	5	9	9	16	5	6
R1	Jääkukka	115 g	64	85	44	52	70	73	68	84	67	38	70	24
R2		275 g	21	13	10	4	9	6	8	9	7	19	7	5
S1	Jänkhäkunkku	115 g	47	29	23	30	63	57	55	20	50	39	40	43
S2		275 g	4	7	0	11	15	7	15	0	20	7	1	10
T1	Vadelmapusu	115 g	66	62	34	34	82	83	58	48	31	40	11	38
T2		275 g	4	3	8	6	9	11	19	6	19	9	8	2
U1	Inkivääri-ihme	115 g	83	48	38	60	90	44	68	49	21	53	59	36
U2		275 g	16	14	4	14	12	10	0	12	15	14	8	13
V	Eukon Lepytys	115 g	0	0	0	0	0	0	0	0	0	0	0	0
W			0	0	0	0	0	0	0	0	0	0	0	0
X1	Ottopoika	80 ml	78	51	33	21	45	39	31	48	71	7	52	31
X2	Guuma Gissa	80 ml	99	66	44	42	76	59	32	18	67	45	61	51
X3	Hirvenpotku	80 ml	90	55	24	39	52	77	52	19	52	25	59	21
X4	Karhunkämmen	80 ml	95	35	34	28	60	51	49	37	68	37	50	26
Y	Saunahunaja	115 g	4	8	11	9	0	10	3	7	0	2	14	9

Picture 35. Excel file part 2.

Koodi	Hunajat	Koko	Braun-schweig	Hanno-ver	Hameln	Soest	Köln	Duis-burg	Müns-ter	Osna-brück	Olden-burg	Bremen	Lübeck	Rostock
TILATTAVAT TAVARAT (kyllä tai ei)														
TT1	Kestokassit (viedään 20 kpl)		Kyllä		Ei			Kyllä		Ei	Ei	Ei	Ei	Ei
TT2	Myytävät kupit (viedään 5 kpl)		Ei		Ei			Ei		Ei	Ei	Ei	Ei	Ei
TT3	Myytävät aamiaislaudat (viedään 5 kpl)		Ei		Ei			Ei		Ei	Ei	Ei	Ei	Ei
TT4	Maistiaistikut (viedään 2 isoa laatikkoa)		Ei		Kyllä			Ei		Ei	Kyllä	Kyllä	Kyllä	Ei
TT5	Esitteet Saksa (viedään 1 laatikko)		Ei		Kyllä			Ei		Ei	Ei	Ei	Ei	Kyllä
TT6	Esitteet Englanti (viedään 10 cm nippu)		Ei		Ei			Ei		Ei	Ei	Ei	Ei	Ei
TT7	Punaiset muovipussit (viedään 1 laatikko)		Ei		Ei			Ei		Ei	Ei	Ei	Kyllä	Ei
TT8	Keltaiset arpalaput (viedään 15cm pino)		Ei		Ei			Ei		Kyllä	Ei	Ei	Ei	Kyllä
TT9	Sellofaanit pieni (viedään 1cm pino)		Ei		Ei			Ei		Kyllä	Ei	Ei	Ei	Ei
TT10	Sellofaanit iso (viedään 0,5cm pino)		Ei		Ei			Ei		Ei	Ei	Ei	Ei	Ei
TT11	Kosteuspyyhkeitä (viedään 2 pakettia)		Ei		Kyllä			Kyllä		Ei	Ei	Ei	Ei	Ei
TT12	Käsipyyhepaperi (viedään 2 pakettia)		Ei		Ei			Kyllä		Kyllä	Kyllä	Ei	Ei	Ei
TT13	Mustat roskapussit (viedään 2 rullaa)		Ei		Ei			Ei		Kyllä	Ei	Ei	Ei	Ei

Picture 36. Excel file sellers equipment parts (APP).

TILA	TTAVAT TAVARAT ja KORJATTAVAA (vapaa tekstikenttä)												
KK1	Muuta myytävää tai lisätietoja invikseen	Ei		Ei			Ei		isoja jätesäkkejä	maistia istikkuja tarvis	Ei	Ei	Ei
KK2	Korjattavaa / mökin varusteissa puutteita	paketoi ntipöydän		Ei			Ei		Ei	Ei	Ei	Ei	Ei
KK3	Muut myyjän tarviketarpeet	Ei.		Ei.			Ei.		ei mittään	Ei.	Ei.	Ei.	Talletu skortti automa
SMS - tarviketilaukset yms.													
SS	Mitä muuta?		kosteus pyyhkeitä käsipyyhepape		Maistia istikkuja lisää	teippiä käsipaperia tikkuja		Ei. (Manual Add)					

Picture 37. Excel file comment part (APP and SMS).

Every time the writing inside the Excel file of the inventory is complete, ExcelManagement save the file. In the case the first inventory written in the file, the system saves the file with the name “Inventory_at_date” where date is date of the inventory.

As last step of this process, the content of the list SmsTemporanyStorage is stored inside the Database for further statistics or data analysis. The function called InsertAppSMS, that is part of the DatabaseManagement class, takes as argument the list SmsTemporanyStorage, it checks the number of elements inside the collection. If the collection has more than 64 elements which means that is an inventory sent as Android application, a long query is prepared (Appendix 2 h). To avoid writing inside the code a large amount of statements, name, phone, date, place are hard coded, for other elements of the inventory, the body with product name and quantity is looped. The temp array contains the column where the data needs to be inserted plus its quantity, the query

is prepared and executed. In the same way, the inventory from normal SMS is inserted in the Database but the only difference is that in the second case, the amount of inventory field is less.

In conclusion, every step described in this chapter is carried out until all the market places are retrieved. Every time an inventory cycle is completed, a variable that counts the number of inventory inbound, is incremented by one. When it arrives to the total number of market place, the Automated Inventory Application invokes the EmailService class that prepares the company Excel file with summary of the inventory as attachment of automated email and send it to the management of the company.

In the next chapter, an analysis of the powerful log system of Automated Inventory is present. Moreover, it describes and discusses all the challenge faced during the implementation of this software.

6 CHALLENGES

The Automated Inventory Application presents different type of challenges. These challenges concern only the server side of the application. This chapter presents the most important of them categorised as follows:

- Challenges in the design of the code
- Human errors in sending normal SMS
- Log File system that supervises all the operation of Automated Inventory Application

6.1 The code design

The communication occurs between the computer and the FONA 3G Module. The connection and the exchange of information happens with the C#, Serial Connection. It takes as parameters the port name to which the modem is connected. To get this string automatically, without looking every time in the device manager of windows, the class Management was written making the port available to the whole program. This was crucial because every time the modem needs to exchange information with Automated Inventory Application, a serial connection port is established and it is closed when the exchange of information finish. The conversion of the SMS inbound from Unicode format to GSM format was a great challenge. The problem was that during the automatic conversion, the Finnish characters were not recognised, and a question mark was appended instead of the character. The function to do this job is called “ConvertUCS2toReadableFormat”, (Picture 38) and takes as parameter the message in hexadecimal format and “translates it” in human readable format.

```
private string ConvertUCS2toReadableFormat(string message)
{
    List<byte> bytes = new List<byte>();

    for (int i = 0; i < message.Length; i += 2)
    {
        bytes.Add(byte.Parse(message.Substring(i, 2), System.Globalization.NumberStyles.HexNumber));
    }
    //get the message converted
    var messageConverted = Encoding.BigEndianUnicode.GetString(bytes.ToArray());

    return messageConverted;
}
```

Picture 38. Conversion from Unicode to GSM charset format.

The bytes added each time are recognised as hexadecimal number. The challenge here was to understand that the encoding type from of the SMS had Big Endian Unicode characters. Big or Small Endian is the way which the multibyte are organized, for example, zero could be represent as 0x00 or 0xFF. In the case of Finnish characters, the SMS was encoded in Big Endian Unicode.

6.2 Human errors

The human errors occur typically when the inventory is sent via SMS. The Android Application has already a system that checks if all the data stored is correct, and it is impossible to proceed if the data inserted is wrong. In the normal SMS case, if the sales agents made mistake in completing the SMS form, the software saves the values without any number, or in the worst cases, the message was truncated and only half-saved. The solution to avoid this problem was the implementation of a function called “watchdog” that take as parameter all the message collected in the ListBuilder list and check if all the necessary content is inside.

The “watchdog” function (Appendix 2 i) which is present in the SMS class pursues two objectives, firstly one to check if all the elements inside that must be in the list are present. The second objective is to check if all the parts of the SMS or the APP have arrived and are read by the modem and inserted in the list. If one of the two conditions is not applied, the application assigns null to the list and a new cycle needs to be repeated. The watchdog function works by looping every element of the list that contains all the SMS body, checking if all the product names and quantity are there. A warning message shows when the code of the software arrives to this point and when the check is complete. If the response is not positive, the watchdog function shows which element is missing. It shows also if the problem is in the SMS or in the APP. When the watchdog function finds a mistake, the id, the serverity of the event raises to 10 meaning that the message returns null and a new cycle is called (Picture 39).

10	21.31.08 ip.	SMS class - Watchdog	The message (SMS) doesn't contain X1
0	21.31.08 ip.	SMS Class - WatchDog Function	WatchDog End

Picture 39. Extract of the log file.

When the Watchdog function was implemented, all the errors related to this kind of problem were resolved.

Another problem found in the normal SMS part, was that sometimes the sales agents put the data in a different way that the one required. For example, some sales agents omit the year, throwing an exception in the main program because the software was programmed to read the date in a certain manner. Another problem was when the sales agents omit to write their name or surname, in this case the date was inserted as the surname of the sale agent. The solution of this problem was to insert a “Date try parse” for the element of the array that should be contain the date, and fix the surname and date array position by repositioning these elements.

```
DateTime date1;
if ((System.DateTime.TryParseExact(temp[3], "dd'.'mm'.'yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out date1)) || (System.DateTime.TryParseExact(temp[3], "dd/mm/yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out date1)))
{
    name = temp[2].ToString();
    date = temp[3].ToString();
    place = temp[4].ToString();

    if (place.Contains("Dresden"))
    {
        place = temp[4] + " " + temp[5];
        ii = 6;
    }
    ii = 5;
}
else
{
    name = temp[2].ToString() + " " + temp[3].ToString();
    date = temp[4].ToString();
    place = temp[5].ToString();

    if (place.Contains("Dresden"))
    {
        place = temp[5] + " " + temp[6];
        ii = 7;
    }
    ii = 6;
}
```

Picture 40. Statement that verifies the date.

This statement solved the date and surname problem. The first and second time that the inventory was in progress, another mistake in the SMS message was found. Many sales agents forget to insert the word “Ei.” (no) after the SS comment part if they did not have the need to order equipment. This did not generate a huge problem in the software, only an exception in the Database function because the last element was not found and the cell of Excel inventory file relative to SS was empty. The following fix was introduced (Picture 41):

```
if(temp[temp.Length -1].Contains("SS"))
{
    smsTemp.Add(new SmsTemporaryStorage(name, phone, date, place, "SS", "Ei. (Manual Add)"));
    mgt.Warning(0, "SMS Class - ProdandQty function", "Added manually the SS");
```

Picture 41. Manual Add of the missing field (last one).

This simple fix checks the last element of the temp array where all the inventory data, product and quantity are in, and control if the element is SS. This means that nothing is written in there and the comment is forgotten, so the Automated Inventory Application

adds manually “Ei. (Manual Add)” (no (Manual Add)). A warning message shows that the comment in the SMS was missing.

6.3 Log File system

The logfile system has the purpose to register all the events that the Automated Inventory Application generates during its entire process. The class responsible is the FileManagement. The log file has two different tasks:

1. To write a general log file, with all the information and event, from when the program is launched until the program is closed.
2. To it collects the inventory in a log file when the function “SmsProdandQty”, the function that gets the product code and quantity in the SMS class, has finished splitting the strings.

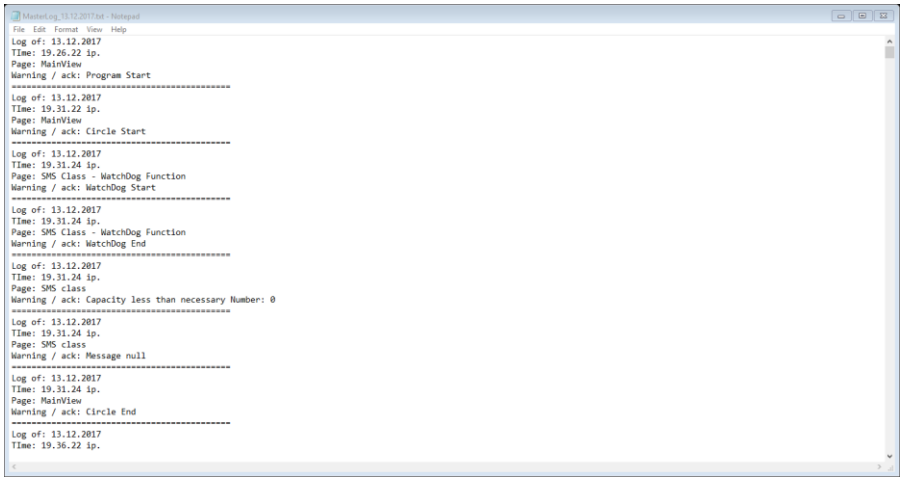
This latter task is achieved using the StringWriter class. When it is called, the class opens a file stream to insert the data inside a text file, and in the end, it closes the file. The general log file is done with the following function:

```
39 references
public void Warning(int id,  string Page, string Warning)
{
    DateTime date = DateTime.Today;
    DateTime now = DateTime.Now;
    CreateMainFolder();
    CreateSubFolder();

    string path = MainDir + "/" + date.ToShortDateString() + "/" + "MasterLog_" + date.ToShortDateString() + ".txt";

    using (StreamWriter logWriter = new StreamWriter(path, append: true))
    {
        try
        {
            logWriter.WriteLine("Log of: " + date.ToShortDateString().ToString());
            logWriter.WriteLine("Time: " + now.ToString("HH:mm:ss tt"));
            logWriter .WriteLine("Page: " + Page);
            logWriter.WriteLine("Warning / ack: " + Warning);
            logWriter.WriteLine("=========================================");

            logWriter.Close();

            App.Current.Dispatcher.Invoke((Action)delegate {

                war.Add(new Warnings(id, now.ToString("HH:mm:ss tt"), Page, Warning)); //ori date.ToLongDateString()

            });
        }
        catch(Exception ex)
        {
            MessageBox.Show("Error in writing the log " + ex.Message, "Automated Inventory Application", MessageBoxButton.OK, MessageBoxImage.Error);
        }
```

Picture 42. Warning function.

This function, takes as arguments the severity of the warning, i.e. identification, id, the page which the class has generated the warnings, i.e. page, and the warning message itself. The path file where the log file is saved, is done with the function “CreateFolder” and “CreateSubfolder” in the same way which the Excel file is created. The log file reports the time which the warning is generated. When an event is handled correctly, it writes it in the log file. When an exception is triggered, the functions reports the class and the

message generated from the exception in this file. The logs are also reported in the MasterLog file thanks to the ObservableCollection that handles all these messages. The final user can simply verify in the windows clicking the dropdown menu management then MarterLog file. All the log files are saved inside a text file in a folder named with the date where the file is generated.



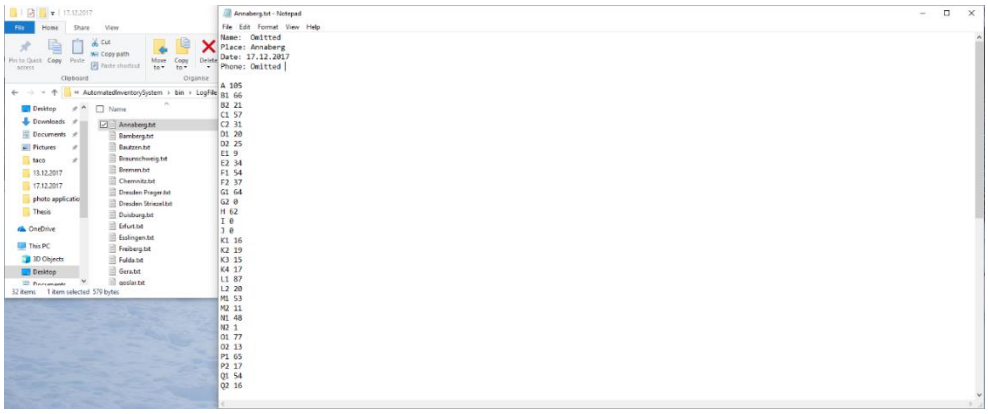
Picture 43. Master-log file in text format.

The second type of log file contains reports for each market, basic information of sale market, name of the sales agents, market name, phone number. It also reports the codes of the products with all the quantity. It saves the interested text file in a subfolder of the program named with the date of the inventory. All the markets have their log file named with the market place name.

```
1 reference
public void WriteLogFile(List<SmsTemporaryStorage> list)
{
    DateTime date = DateTime.Today;

    //string fileName =MainDir + date.ToShortDateString() + "/" + list[0].Identity + " " + list[0].Place;
    string path = MainDir + "/" + date.ToShortDateString() + "/"  + list[0].Place + ".txt";

    using (StreamWriter writer = new StreamWriter(path))
    {
        try
        {
            writer.WriteLine("Name: " + list[0].Identity);
            writer.WriteLine("Place: " + list[0].Place); ;
            writer.WriteLine("Date: " + list[0].Date);
            writer.WriteLine("Phone: " + list[0].Phone);
            writer.WriteLine();

            foreach (SmsTemporaryStorage s in list)
            {
                writer.WriteLine(s.ProductName + " " + s.Qty);
            }

            writer.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error in writing the log " + ex.Message, "Automated Inventory Application", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
}
```

Picture 44. Function that writes product and quantity to a text file.

This objective is achieved writing first the basic information. Instead of inserting all the product codes and quantity, a loop i.e., “foreach loop”, is used. The loop takes all the elements of the list which the product and quantity are saved and write them inside a text file.



Picture 45. Example of log file.

In conclusion, this log file system is studied to double check the operation of the Automated Inventory Application. It is possible to verify that the data inserted in the company’s inventory Excel file is correct. In case of error or exception, it is possible to check and verify the product inserted, just by clicking the interested market.

In the next chapter, the evaluations of this thesis are presented. An analysis of the of the result of the application are given, also the impressions of the sales agents and the management of the company are reported.

7 EVALUATION

Evaluation of the Automated Inventory Application is essential for the future prospects and conclusion. In this chapter the results are being evaluated from the both sides of the application:

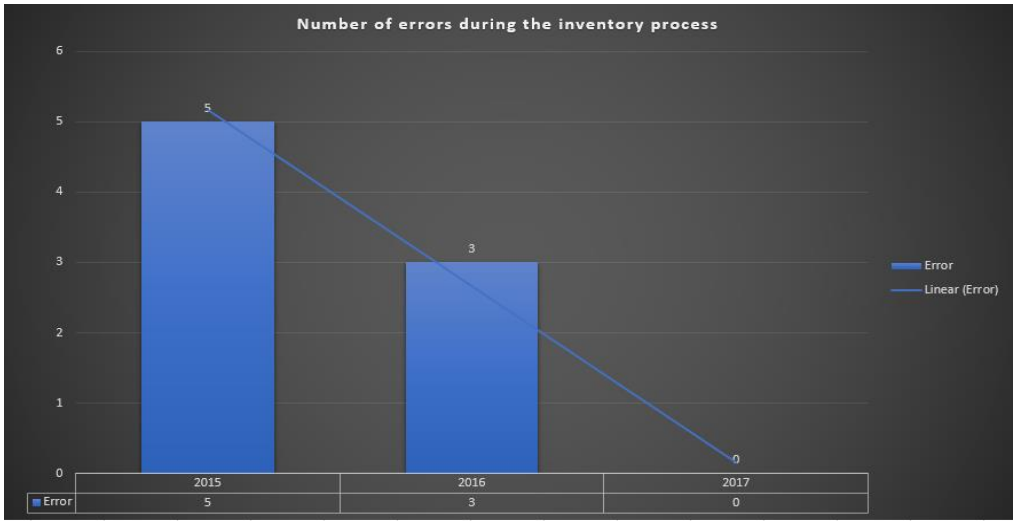
- Server side
- Client side

The discussion focuses on the differences between the past method of the company and the new method which includes the Application that has been implemented. The data of the previous years is taken from the working hours of the author of this thesis. In addition, sales agent reports and discussions with the members of the management are taken in consideration.

Note that the Automated Inventory Application was tested with a German SIM card inside Finland. When it received the inventory messages from Germany with a German SIM card, the messages arrived late, sometimes with five minutes delay. This problem was related to some delay in the international phone lines. However, when the application was used in Germany the problem written above was absent and fewer problems also appeared when sales agents were using Finnish SIM cards to send messages. To avoid this problem in the future, the commissioner has decided to use in the future only Finnish SIM cards in Germany.

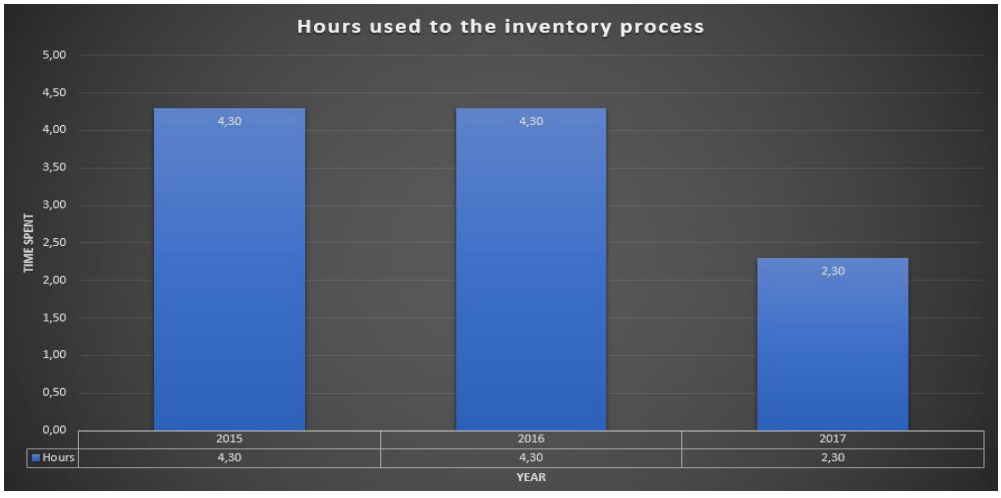
7.1 Server side

Before the introduction of the Automated Inventory Application, the SMS were manually copied from the phone screen to an Excel file that summarized the inventory. After the introduction of this application, the number of errors dropped to zero as demonstrated in the following graph:



Picture 46. Analysis of number of error in past year inventory.

When the inventory was completed with the traditional manual method, on average, depending when the data was taken, some errors in copying were always found, in 2015 an average of five, and in 2016 about three errors. The graph clearly demonstrates that heaving the application doing the process of inventory has minimalized errors. The application does not make mistakes, because the loop takes all the elements of the inventory and appends the numbers in the company Excel file. Another positive result is the time that the application uses to complete all the process.

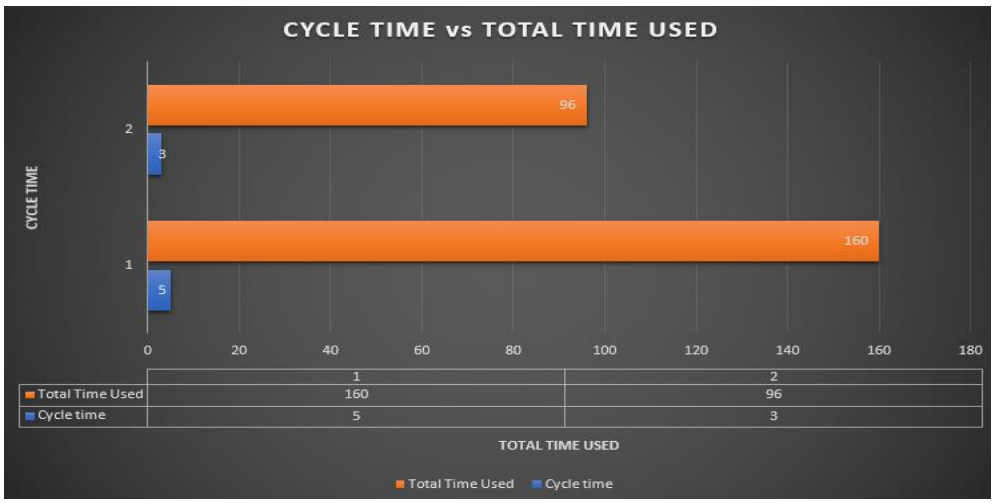


Picture 47. Time spent in the inventory process.

Before the introduction of the application, the time spent to complete an inventory was about four and half hours, because writing all the numbers inside the company file was

a long process, and the SMS always arrived during a long period of time. The Automated Inventory Application takes about five to seven seconds to complete the inventory of one market place. With the traditional method, the time spent on one market place was longer, because first all the numbers were handwritten inside the Excel file and after that they were checked to avoid mistakes. The Automated Inventory Application was tested with a cycle of five minutes and three minutes. As mentioned already in Chapter 4, the cycle time represents how much time the application waits before it asks the modem to retrieve the SMS inside its memory.

The following results were achieved:



Picture 48. Different cycle of time assigned to the function do_inventory.

Because the application was first time used in practise, it was always supervised while running, because it was still in testing phase. In the future, when the application will be completely reliable, it can be left alone and it does not need a supervisor and the time used to supervise will be spent in other operations. Basically it takes two minutes to check if the operation of inventory is completed successfully, especially when all the sales agents will use the Android application instead of the normal SMS.

7.2 Client side

The Android Application was working as expected being adaptive to all Android versions above five. The graphical user interface is user-friendly, and responsive with different phone sizes. As reported from the sales agents, the application never gave any errors

or exceptions. However, one problem was discovered with a sales agent who used a phone bought from England, with English SIM and with Android five. For some reasons, i.e., different type of encoding and the English message center dispatcher, the messages arrived with a faulty representation of Finnish letters. The solution was to let the sales agent to use the traditional SMS method.

The sales agents who were using Android Application were interviewed and the feedback has been extremely positive. Some sales agents said that with the new application the inventory process became almost fun, because writing a traditional SMS message with an old phone was boring. They outlined three major points. The first one was that they did not make mistakes in forgetting a product or a quantity, the second one was that they always double checked needs for other equipment as these need were now asked separately with questions “yes” or “no” and there were no possibility to continue without replying. The third point was that the inventory process was now more fun and easy.

In case of the traditional SMS, the application was able to correct some human errors, but it is still much more unreliable and sensitive than the Android application. Therefore, in the future, all the sales agents should use the Android application. However, it is important to have a backup system, for example, touchscreen phones are easy to break. In this case, it is possible to buy a twenty euro phone, and use the traditional SMS system.

In the next Chapter, conclusions are proposed.

8 CONCLUSION

The Automated Inventory Application has demonstrated how it is possible to automatize a process of inventory in a small company. This application has automatized the process that the company had done manually for about fourteen years. The main objective achieved were the reduction of errors and time spent in this process.

It is part of a modern business intelligence to think, how even a small company can automatize their old processes and leave more time for other tasks. With the help of computer engineering this can be achieved and it gives the company better aspects for growth. It can also give the company an advantage in the competitive situation and increase the attractiveness in the eyes of the employees. The sales agents are young adults and they have grown up with current technology therefore for them for example, using an Android telephone is more normal than sending a SMS message.

Even though the Automated Inventory Application has been tested already succesfully, it is still not ready and it will probably never be, because the technology changes all the time and new ideas come up every year. The product data needs to be changed yearly and the application could be also used for sending some other information, not only inventory. Also some future upgrades have been thought already during last season.

The core of the Automated Inventory Application is complete. To make this system better, and more precise, a series of new functions will be implemented. An internal console is already in development phase. The internal terminal is a window where is possible to write command directly for the modem. Now the user of the Automated Inventory Application server side uses “Putty” a third-party software specialized in these types of communications. It is better to have a system like an internal console because every time a communication is needed, the Automated Inventory Application needs to be shut down, open the third-party software software, and establish connection with it.

A more sophisticated method to avoid repetition of the SMS should be also implemented, in the situation when for example, there are network problems and for some reason sales agent’s inventory comes twice. This should not produce exception, but the Automated Application should correct it automatically.

Next Christmas season, instead of the local database, Microsoft Azure should be used. This is very important because Azure offers a service of exchange data between the

online database and the mobile phone. This year, the Android Application can recognise only if a number is outside the range of the inventory and the maximum amounts of products is theoretically, discovered by making statistics between different year. With Azure, it will be possible to consult the online database and check if the number reported to the sales agent is real, because the management will put the amounts of products transported there. This will be possible adding a plugin to the Automated Inventory Application. The plug-in will loads in the database how many new products are taken physically after the inventory to a sale market and every time a new inventory process is filled inside the Android application the program will consult the data. Every time a product is loaded in some market, the program consults the online database and checks if the number is real, more precisely if the number is smaller or greater than the physical product inside the market place.

One of the most important upgrades before the next Christmas period, would be to create the application client side for IOS, the Apple iPhone operating system, because about thirty percent of the sales agents use iPhones. It is clever idea to also cater for those sales agents that use a different operating system. This upgrade is significant also if the company wants to use in the future only touch screen telephones and leave SMS messages only as back-up system. It will also lower costs so that the company doesn't need to buy several Android telephones and sales agents can use their own.

As a conclusion the Automated Inventory Application has given the company a lot to think about, how this type of computer engineering could be utilized even more. This type of applications specifically done for the need of the company, could be sold also for other companies.

REFERENCES

[1] SearchSQLServer (2017) Database(DB). [online] Available at http://searchsqlserver.techtarget.com/definition/database [Accessed 25th September 2017]

[2] Microsoft Docs. (2015) Get Started with .NET Framework. [online] Available at https://docs.microsoft.com/en-us/dotnet/framework/get-started/index [Accessed 8th January 2018].

[3] Microsoft Docs. (2017) Common Language Runtime. [online] Available at https://docs.microsoft.com/en-us/dotnet/standard/clr [Accessed 8th January 2018].

[4] Microsoft Docs. (2017) Classes. [online]. Available at https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/classes [Accessed 8th January 2018].

[5] Stroustrup Bjarne. (2013) The C++ Programming Language fourth edition. New Jersey. Addison-Wesley, pp 449-451.

[6] Stroustrup Bjarne. (2013) The C++ Programming Language fourth edition. New Jersey. Addison-Wesley, pp 455-456.

[7] Microsoft Docs. (2017) Objects (C# Programming Guide) [online]. Available at https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/objects [Accessed 8th January 2018]

[8] Xamarin Inc. (2018) Deliver native Android, iOS, and Windows apps, using existing skills, team and code. [online]. Available at https://www.xamarin.com/platform [Accessed 9th January 2018].

[9] Microsoft Docs. (2015) Introduction to the C# Language and the .NET Framework [online] Available at https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework [Accessed 8th January 2018].

[10] Skeet J. (2014) C# in depth third edition. New York. Manning Publications Co, pp. 416-419.

[11] MSDN (2017) Serial Port Class. [online]. Available at https://msdn.microsoft.com/en-gb/library/system.io.ports.serialport(v=vs.110).aspx [Accessed 9th January 2018].

[12] The Linux Information Project (2005) [online]. Available at http://www.linfo.org/at_command_set.html [Accessed 9th January 2018].

[13] Gamrel, B. (2012). Database Administration Fundamentals. [PDF]. Hoboken, NJ. pp 2-12.

[14] MSDN (2017) Sql Connection Class. [online]. Available at https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection(v=vs.110).aspx [Accessed 9th January 2018].

[15] WPF Tutorial (2018) What is WPF? [online]. Available at http://www.wpf-tutorial.com/about-wpf/what-is-wpf/ [Accessed 9th January 2018].

[16] Visual Studio Docs (2016) Introduction to WPF [online]. Available at https://docs.microsoft.com/en-us/visualstudio/designers/introduction-to-wpf [Accessed 9th January 2018].

[17] Microsoft Docs. (2016) Managed Execution Process. [online]. Available at https://docs.microsoft.com/en-us/dotnet/standard/managed-execution-process [Accessed 9th January 2018].

[18] Technopedia (2017) Modem [online]. Available at https://techterms.com/definition/modem [Accessed 9th January 2018].

[19] Poole, I. Radio-Electronics. [online] Available at http://www.radio-electronics.com/info/cellulartelecomms/gsm_technical/gsm_architecture.php [Accessed 9th January 2018].

[20] Poole, I. Radio-Electronics. [online] Available at http://www.radio-electronics.com/info/cellulartelecomms/gprs/gprs_tutorial.php [Accessed 9th January 2018].

[21] Adafruit Webpage(2017) Adafruit FONA 3G Cellular + GPS Breakout. [online] Available at https://learn.adafruit.com/adafruit-fona-3g-cellular-gps-breakout/overview [Accessed 20th September 2017]

[22] Skeet J. (2014) C# in depth third edition. New York. Manning Publications Co, p. 20.

[23] Developers home. (2017) Introduction to AT commands. [online] Available at http://www.developershome.com/sms/atCommandsIntro.asp [Accessed 25th September 2017]

[24] Extensive Markup Language (XML) (2016). [online] Available at http://www.w3.org/XML/ [Accessed 25th October 2017]

[25] Microsoft Docs. (2017). Using the StringBuilder Class in .NET. [online] Available at https://docs.microsoft.com/en-us/dotnet/standard/base-types/stringbuilder [Accessed 25th October 2017]

[26] Developers. (2017) [online] Available at https://developer.android.com/reference/android/telephony/SmsManager.html [Accessed 26th October 2017]

[27] Developers. (2017). [online] Available at https://developer.android.com/reference/android/content/Intent.html [Accessed 25th October 2017]

Automated Application Client side

```
1 reference
private void LogInProcedure(EditText user, EditText psw)
{
    user = FindViewById<EditText>(Resource.Id.user);
    psw = FindViewById<EditText>(Resource.Id.psw);

    string username = user.Text;
    string password = psw.Text;
    string accountUsr = "test";
    string accountpsw = "test";

    //check if the string are empty (user and psw) {1}
    if(string.IsNullOrEmpty(username) || (string.IsNullOrEmpty(password)))
    {
        Toast toast = Toast.MakeText(this, "Fill all the field please", ToastLength.Long);
        toast.SetGravity(Android.Views.GravityFlags.Top, 0, 0);
        toast.Show();
    }
    //check if user and psw are ok {2}
    if((username == accountUsr) && (password == accountpsw))
    {
        Toast toast = Toast.MakeText(this, "Login ok ", ToastLength.Long);
        toast.SetGravity(Android.Views.GravityFlags.Top, 0, 0);
        toast.Show();
        StartActivity(typeof(Information)); // go to the next screen
    }
    else //wrong {3}
    {
        Toast toast = Toast.MakeText(this, "Username or password wrong ", ToastLength.Long);
        toast.SetGravity(Android.Views.GravityFlags.Top, 0, 0);
        toast.Show();
    }
}
```

Appendix a Log in procedure function.

```
1 reference
private void GetDateForUser(EditText day)
{
    day = FindViewById<EditText>(Resource.Id.day);
    DateTime time = DateTime.Now;
    day.Text = time.Date.ToString("dd/MM/yyyy");
}
```

Appendix b Insert date function.

```
//create a dropdown menu {1}
1 reference
private void DropDownMenu(Spinner spinner)
{
    spinner = FindViewById<Spinner>(Resource.Id.spinner);
    spinner.ItemSelected += new EventHandler<AdapterView.ItemSelectedEventArgs>(Selection);

    var adapter = ArrayAdapter.CreateFromResource(this, Resource.Array.place_array, Android.Resource.Layout.SimpleSpinnerItem);
    adapter.SetDropDownViewResource(Android.Resource.Layout.SimpleSpinnerItem);
    spinner.Adapter = adapter;
}
```

Appendix c: Dropdaown menu.

```
//fill the list
1 reference
void FillTheList()
{
    //get element
    day = FindViewById<EditText>(Resource.Id.day);
    name = FindViewById<EditText>(Resource.Id.name);
    // if same elements are null or empty
    if((string.IsNullOrEmpty(day.Text)) || (string.IsNullOrEmpty(name.Text)) || (TempLocation == "Valitse paikka"))
    {
        Toast toast = Toast.MakeText(this, "Fill all the field please", ToastLength.Long);
        toast.SetGravity(Android.Views.GravityFlags.Top, 0, 0);
        toast.Show();
    }
    else
    {

        //clean the list
        sellerInfos.Clear();
        //store it
        //non static method
        sellerInfos.Add(name.Text.ToString()); //{1}
        sellerInfos.Add(day.Text.ToString());
        sellerInfos.Add(TempLocation.ToString());

        //build the head of the message
        MessageBuilder smsBuilder = new MessageBuilder(); //{2}
        smsHead = smsBuilder.CreateTheSMSHead(sellerInfos);


        //Create the intent for the next activity store the message for the next
        var luonollinenActivity = new Intent(this, typeof(luonnollinen)); //{3}
        luonollinenActivity.PutExtra("smsHead", smsHead);
        StartActivity(luonollinenActivity);

    }
}
```

Appendix d: Fill the list function.

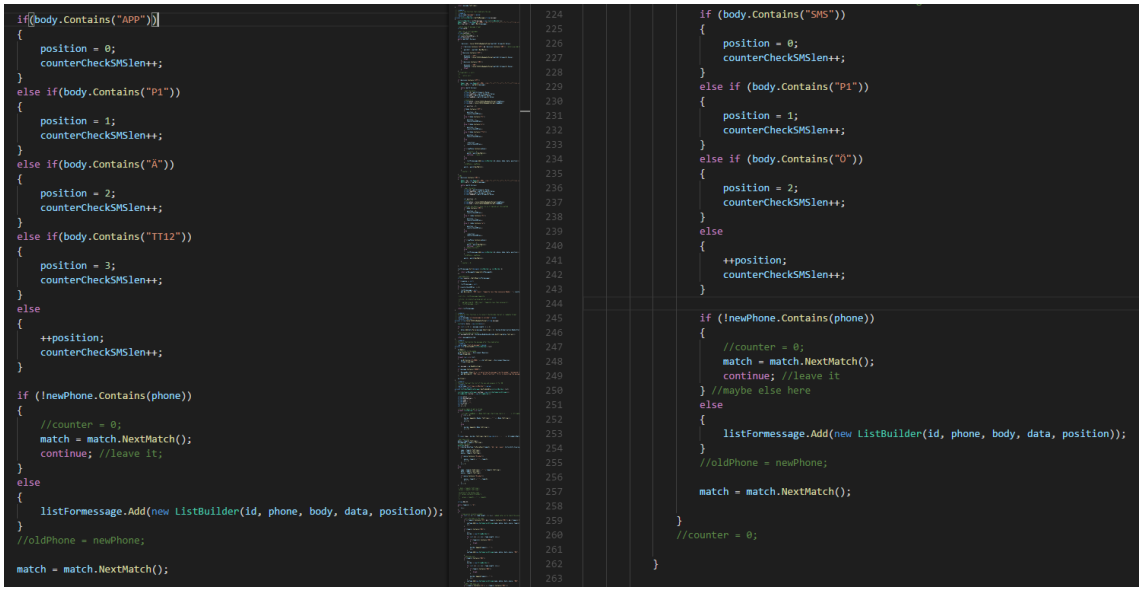
```
1 reference
private void DoSinappiInventory()
{
    //{1}
    string[] SinappiName = new string[] {"X1", "X2", "X3", "X4", "Y", "Z", "Å", "Ä", "Ö" };

    int index = 0;

    List<EditText> Element = new List<EditText>();//{2}
    List<BodyInventory> sinappiList = new List<BodyInventory>(); //{3}

    //{4}
    EditText X1 = FindViewById<EditText>(Resource.Id.X1);
    EditText X2 = FindViewById<EditText>(Resource.Id.X2);
    EditText X3 = FindViewById<EditText>(Resource.Id.X3);
    EditText X4 = FindViewById<EditText>(Resource.Id.X4);
    EditText Y = FindViewById<EditText>(Resource.Id.Y);
    EditText Z = FindViewById<EditText>(Resource.Id.Z);
    EditText SvedA = FindViewById<EditText>(Resource.Id.svedA);
    EditText SvedO = FindViewById<EditText>(Resource.Id.svedO);
    EditText SvedOO = FindViewById<EditText>(Resource.Id.svedOO);


    Element.Add(X1);
    Element.Add(X2);
    Element.Add(X3);
    Element.Add(X4);
    Element.Add(Y);
    Element.Add(Z);
    Element.Add(SvedA);
    Element.Add(SvedO);
    Element.Add(SvedOO);
```

Appendix d: Basic declaration of the function inventory.

Automated Inventory Application Server side

```
1 reference
public void SendSMS(string phone, string text)
{
    try
    {
        sp = new SerialPort(); //serial port declaration
        sp.PortName = port.ToUpper().ToString();     //port name
        sp.Open();
        sp.WriteLine("AT" + Environment.NewLine); //AT cmd set AT ->ok
        Thread.Sleep(100);
        sp.WriteLine("AT+CMGF=1" + Environment.NewLine); //set the net to message mode
        Thread.Sleep(100);
        sp.WriteLine("AT+CSCS=\"GSM\"" + Environment.NewLine); //check if the net support sms
        Thread.Sleep(100);
        sp.WriteLine("AT+cmgs=\"" + phone.ToString() + "\"" + Environment.NewLine); //put the number
        Thread.Sleep(100);
        sp.WriteLine(text.ToString() + Environment.NewLine); //send the text
        Thread.Sleep(100);
        sp.Write(new byte[] { 26 }, 0, 1); //write everything
        Thread.Sleep(100);

        var response = sp.ReadExisting(); //the response from the GSM net

        if (response.Contains("ERROR"))  //handling error
        {
            MessageBox.Show("Error", "Error", MessageBoxButton.OK, MessageBoxImage.Asterisk);
            mgt.Warning(0, "Sending SMS", "Problem in sending SMS");
        }
        else
        {
            MessageBox.Show("done, Message Sent", "Message sent", MessageBoxButton.OK, MessageBoxImage.Asterisk);
            mgt.Warning(0, "Sending SMS", "Message Sent");
        }
        sp.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error in Sending SMS", MessageBoxButton.OK, MessageBoxImage.Error);
        mgt.Warning(0, "Sending SMS", "Problem in sending SMS");
    }

}
```

Appendix e Function that send SMS.



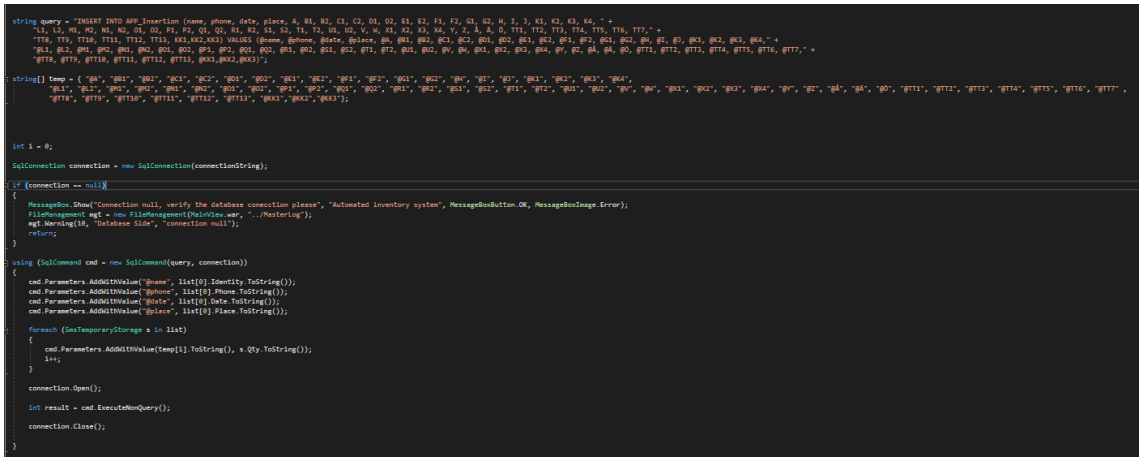
```
if(body.Contains("APP"))
{
    position = 0;
    counterCheckSMSlen++;
}
else if(body.Contains("P1"))
{
    position = 1;
    counterCheckSMSlen++;
}
else if(body.Contains("Ä"))
{
    position = 2;
    counterCheckSMSlen++;
}
else if(body.Contains("TT12"))
{
    position = 3;
    counterCheckSMSlen++;
}
else
{
    ++position;
    counterCheckSMSlen++;
}

if (!newPhone.Contains(phone))
{
    //counter = 0;
    match = match.NextMatch();
    continue; //leave it;
}
else
{
    listFormessage.Add(new ListBuilder(id, phone, body, data, position));
}
//oldPhone = newPhone;

match = match.NextMatch();
```

```
            if (body.Contains("SMS"))
            {
                position = 0;
                counterCheckSMSlen++;
            }
            else if (body.Contains("P1"))
            {
                position = 1;
                counterCheckSMSlen++;
            }
            else if (body.Contains("Ö"))
            {
                position = 2;
                counterCheckSMSlen++;
            }
            else
            {
                ++position;
                counterCheckSMSlen++;
            }

            if (!newPhone.Contains(phone))
            {
                //counter = 0;
                match = match.NextMatch();
                continue; //leave it
            } //maybe else here
            else
            {
                listFormessage.Add(new ListBuilder(id, phone, body, data, position));
            }
            //oldPhone = newPhone;

            match = match.NextMatch();

        }
        //counter = 0;

    }
```

Appendix f Matching Keywords for APP and SMS.

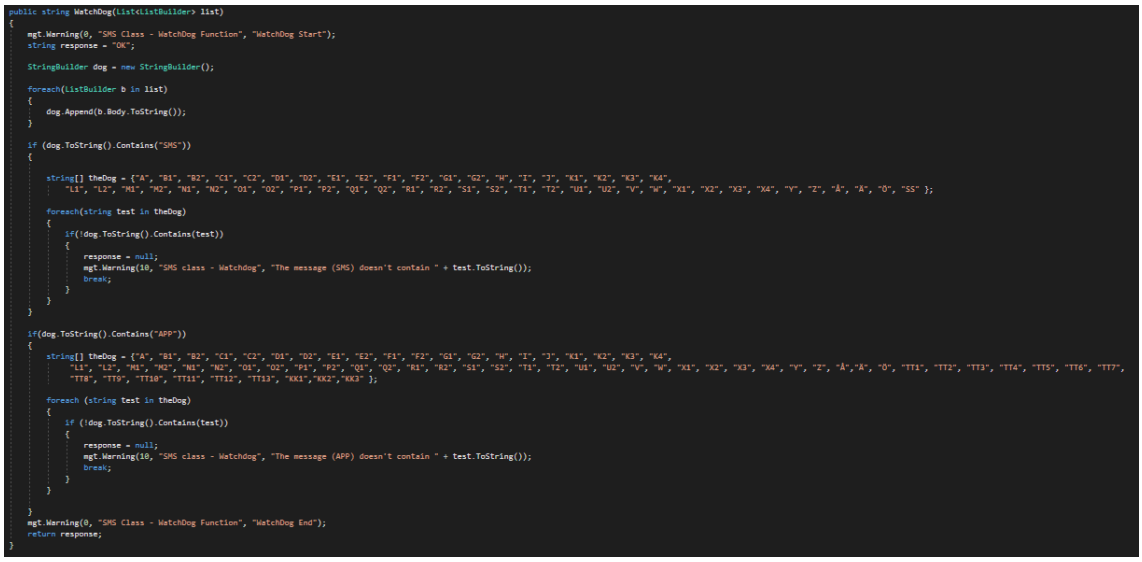
```
3 references
public void CreateMainFolder()
{
    try
    {
        bool isThere = Directory.Exists(MainDir);
        //if the folder doesn't exist
        if (!isThere)
        {
            Directory.CreateDirectory(MainDir);
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(messageBoxText: "Impossible to create the main folder for the log " + ex.Message, caption: "AutomatedInventorySystem", button: MessageBoxButton.OK, icon: MessageBoxImage.Error);
    }
}
/// <summary>
/// Create a daily sub-directory
/// </summary>

3 references
public void CreateSubFolder()
{
    try
    {
        DateTime date = DateTime.Today;
        string directoryName = date.ToShortDateString();

        bool IsThere = Directory.Exists(MainDir);

        if(IsThere)
        {
            Directory.CreateDirectory(Path.Combine(MainDir, directoryName));
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(messageBoxText: "Impossible to create the sub-folder for the log " + ex.Message, caption: "AutomatedInventorySystem", button: MessageBoxButton.OK, icon: MessageBoxImage.Error);

    }
}
```

Appendix g FileManagement class creation of folder and subfolder.



```
string query = "INSERT INTO APP_Insertion (name, phone, date, place, A, B1, B2, C1, C2, D1, D2, E1, E2, F1, F2, G1, G2, H, I, J, K1, K2, K3, K4, " +
    "L1, L2, M1, M2, N1, N2, O1, O2, P1, P2, Q1, Q2, R1, R2, S1, S2, T1, T2, U1, U2, V, W, X1, X2, X3, X4, Y, Z, Å, Ä, Ö, TT1, TT2, TT3, TT4, TT5, TT6, TT7," +
    "TT8, TT9, TT10, TT11, TT12, TT13, KK1,KK2,KK3) VALUES (@name, @phone, @date, @place, @A, @B1, @B2, @C1, @C2, @D1, @D2, @E1, @E2, @F1, @F2, @G1, @G2, @H, @I, @J, @K1, @K2, @K3, @K4," +
    "@L1, @L2, @M1, @M2, @N1, @N2, @O1, @O2, @P1, @P2, @Q1, @Q2, @R1, @R2, @S1, @S2, @T1, @T2, @U1, @U2, @V, @W, @X1, @X2, @X3, @X4, @Y, @Z, @Å, @Ä, @Ö, @TT1, @TT2, @TT3, @TT4, @TT5, @TT6, @TT7," +
    "@TT8, @TT9, @TT10, @TT11, @TT12, @TT13, @KK1,@KK2,@KK3)";

string[] temp = { "@A", "@B1", "@B2", "@C1", "@C2", "@D1", "@D2", "@E1", "@E2", "@F1", "@F2", "@G1", "@G2", "@H", "@I", "@J", "@K1", "@K2", "@K3", "@K4",
        "@L1", "@L2", "@M1", "@M2", "@N1", "@N2", "@O1", "@O2", "@P1", "@P2", "@Q1", "@Q2", "@R1", "@R2", "@S1", "@S2", "@T1", "@T2", "@U1", "@U2", "@V", "@W", "@X1", "@X2", "@X3", "@X4", "@Y", "@Z", "@Å", "@Ä", "@Ö", "@TT1", "@TT2", "@TT3", "@TT4", "@TT5", "@TT6", "@TT7" ,
        "@TT8", "@TT9", "@TT10", "@TT11", "@TT12", "@TT13", "@KK1","@KK2","@KK3"};



int i = 0;

SqlConnection connection = new SqlConnection(connectionString);

if (connection == null)
{
    MessageBox.Show("Connection null, verify the database conection please", "Automated inventory system", MessageBoxButton.OK, MessageBoxImage.Error);
    FileManagement mgt = new FileManagement(MainView.war, "../MasterLog");
    mgt.Warning(10, "Database Side", "connection null");
    return;
}

using (SqlCommand cmd = new SqlCommand(query, connection))
{
    cmd.Parameters.AddWithValue("@name", list[0].Identity.ToString());
    cmd.Parameters.AddWithValue("@phone", list[0].Phone.ToString());
    cmd.Parameters.AddWithValue("@date", list[0].Date.ToString());
    cmd.Parameters.AddWithValue("@place", list[0].Place.ToString());

    foreach (GmsTemporaryStorage s in list)
    {
        cmd.Parameters.AddWithValue(temp[i].ToString(), s.Qty.ToString());
        i++;
    }

    connection.Open();

    int result = cmd.ExecuteNonQuery();

    connection.Close();

}
```

Appendix h The query that inser the products and quantity inside the database.



```
public string WatchDog(List<ListBuilder> list)
{
    mgt.Warning(0, "SMS Class - WatchDog Function", "WatchDog Start");
    string response = "OK";

    StringBuilder dog = new StringBuilder();

    foreach(ListBuilder b in list)
    {
        dog.Append(b.Body.ToString());
    }

    if (dog.ToString().Contains("SMS"))
    {

        string[] theDog = {"A", "B1", "B2", "C1", "C2", "D1", "D2", "E1", "E2", "F1", "F2", "G1", "G2", "H", "I", "J", "K1", "K2", "K3", "K4",
            "L1", "L2", "M1", "M2", "N1", "N2", "O1", "O2", "P1", "P2", "Q1", "Q2", "R1", "R2", "S1", "S2", "T1", "T2", "U1", "U2", "V", "W", "X1", "X2", "X3", "X4", "Y", "Z", "Å", "Ä", "Ö", "SS" };

        foreach(string test in theDog)
        {
            if(!dog.ToString().Contains(test))
            {
                response = null;
                mgt.Warning(10, "SMS class - Watchdog", "The message (SMS) doesn't contain " + test.ToString());
                break;
            }
        }
    }

    if(dog.ToString().Contains("APP"))
    {
        string[] theDog = {"A", "B1", "B2", "C1", "C2", "D1", "D2", "E1", "E2", "F1", "F2", "G1", "G2", "H", "I", "J", "K1", "K2", "K3", "K4",
            "L1", "L2", "M1", "M2", "N1", "N2", "O1", "O2", "P1", "P2", "Q1", "Q2", "R1", "R2", "S1", "S2", "T1", "T2", "U1", "U2", "V", "W", "X1", "X2", "X3", "X4", "Y", "Z", "Å","Ä", "Ö", "TT1", "TT2", "TT3", "TT4", "TT5", "TT6", "TT7",
            "TT8", "TT9", "TT10", "TT11", "TT12", "TT13", "KK1","KK2","KK3" };

        foreach (string test in theDog)
        {
            if (!dog.ToString().Contains(test))
            {
                response = null;
                mgt.Warning(10, "SMS class - Watchdog", "The message (APP) doesn't contain " + test.ToString());
                break;
            }
        }

    }
    mgt.Warning(0, "SMS Class - WatchDog Function", "WatchDog End");
    return response;
}
```

Appendix i The watchdog function