Saimaa University of Applied Sciences
Faculty of Technology Lappeenranta
Double Degree
Information Technology

Bronislav Draška, Jakub Stonawski, Ladislav Barabáš

Job-Ticket system for Tehohydro company

Thesis 2013

Abstract

Bronislav Draška, Jakub Stonawski, Ladislav Barabáš
Job-Ticket system for Tehohydro company, 69 pages
Saimaa University of Applied Sciences
Faculty of Technology, Lappeenranta
Information Technology, Double Degree
Thesis 2013
Instructor: Lecturer Pasi Tiihonen, Saimaa University of Applied Sciences

This bachelor thesis discusses a technical task, which was connected with the real company located in Lappeenranta. The objective of the work was to design and create a new information system used for management and administration inside the company. The idea about implementation of the new system was a response to the old one, which had several improvement needs. The work was commissioned by Tehohydro company.

The system has been developed with the help of Microsoft technologies. The development environment was Microsoft Visual Studio 2010 and the programming language was C# using Microsoft .NET Framework 4.0. The database was designed and created in Microsoft SQL Management Studio. All knowledge and information about these technologies was mostly gathered from previous experience during studies in the home university, from real practice work in a company and from the internet.

The final result of this project was a fully functional information system, which has been the subject of numerous testing and subsequently has been deployed into the praxis. Many meetings with the customer in the phase of development ensured that all necessary functionality is included. The work can also be used as a task for the future students, because it may, for example, be extended by a web interface.

Keywords: C#, .NET, SQL, Information system, Design pattern

List of terms

IT	Information technology
CRM	Customer Relationship Management
DBMS	Database Management System
ORM	Object Relational Mapping
MS	Microsoft
C#	Programming language C#
.NET Framework	Software framework developed by Microsoft
JAVA	Programming language JAVA
SQL	Structured Query Language
T-SQL	Transact-SQL
TFS	Team Foundation Server
DP	Design pattern
DO	Data object
ID	Identifier
GUI	Graphical User Interface
IDE	Integrated Development Environment
PDF	Portable Document Format
MD5	Message-Digest
RAM	Random-access memory
GB	Gigabyte
VBA	Visual Basic for Applications
1.NF	First Normal Form
LAN	Local Area Network
DLL	Dynamic-link library

Table of contents

1 Introduction

Internal information systems are a very important aspect for a correct operation of a company at present. Every company has its own management and administration system which is necessary for the function of a whole company. That kind of system was in the past presented in a paper form, where all important data has been written by hand and stored in a card file. But with coming of new technologies all data have been stored on servers by using information systems. This technology can save much more time and work as well, but on the other hand if an information system is not designed correctly the problems may appear. And about this case deals this thesis. An opportunity was offered by Tehohydro company, whose information system had a few unsolved tasks. The system had several improving needs like database handling so if the team would want to improve anything, it would mean that all corporate data would be deleted. Instead of this solution, it was suggested to create a new system with new architecture, all functionality from the previous system and many improvements, which were discussed on the meetings with employees of the company.

The thesis describes the architecture, functionality and database of the created information system called Job-Ticket. All technologies used in development are mentioned as well. This document should serve for students or developers for better understanding of this application from a viewpoint of implementation.

The first chapter explains conditions for deciding to create a new system and what reason led to this decision. The job in the team is described as well and what specialization each member of the team had during the whole development.

The second and third chapters describe basic information about Tehohydro company and their current system, which is actually used. A reader is familiar with technologies, which were used to implementation. There are also advantages and disadvantages in the current system, which were discovered. This part should explain what exactly the system deals and what the main utilization is.

The next chapter contains detailed description of the process during implementation of the created system. It is the most important part, where the architecture with designed diagrams, the database schema with description of linear entities, the functionality of the system and its solutions and finally the phase of the development – testing and deployment are explained. It also explains how data from the old system was transferred into the new one.

The last chapter contains the team’s personal review, what this work gave and what the dealing with customer gave to the team. The work will still continue with the project after submission of this document so there is also one small passage that describes the rest of programming matters, which needs to be done before the final deployment finished version into praxis.

2 About the Job-Ticket project

Everything started in autumn semester 2012. At that time several local companies were visited and the team looking for IT (Information technology) cases for its bachelor thesis. One of these was Tehohydro company that was finally chosen. At the beginning, a couple of possible IT issues that team could solve for them, were obtained for example sugar CRM (Customer Relationship Management). Than all issues were analyzed and finally an issue related to Job-Ticket system was chosen based on discussion with Tehohydro. This issue was about improvement functions of this system, creating new functions and troubleshooting errors.

Unfortunately, during analysis of that Job-Ticket system many aspects how to improve the old system were discovered. Due to those facts the team offered to Tehohydro company to create a new system with better technology, without all these mistakes and with many other advantages which are described later in this thesis.

Team work

The team consists of three members with different experiences and skills in different areas of IT. This fact was used as much as possible so every member was responsible for the area that he understood better than the others. Here is a division of the work on implementation:

- **Jakub Stonawski** was responsible for testing the user interface, its behavior as well as behavior and stability of the whole application. He was also working on many support staff like old data transferring for testing, password handling or checking connection to database server and so on.
- **Ladislav Barabáš** was the main graphic designer. He took care of the look of user interface and its design style as well as of icons or images. Generating of PDF (Portable Document Format) file for printing the job-ticket or sending via e-mail was also completely under his management.

- **Bronislav Draška** designed application architecture, implemented Application logic layer and Database layer of the application. He also took care of the database configuration as well as configuration of the whole database server.

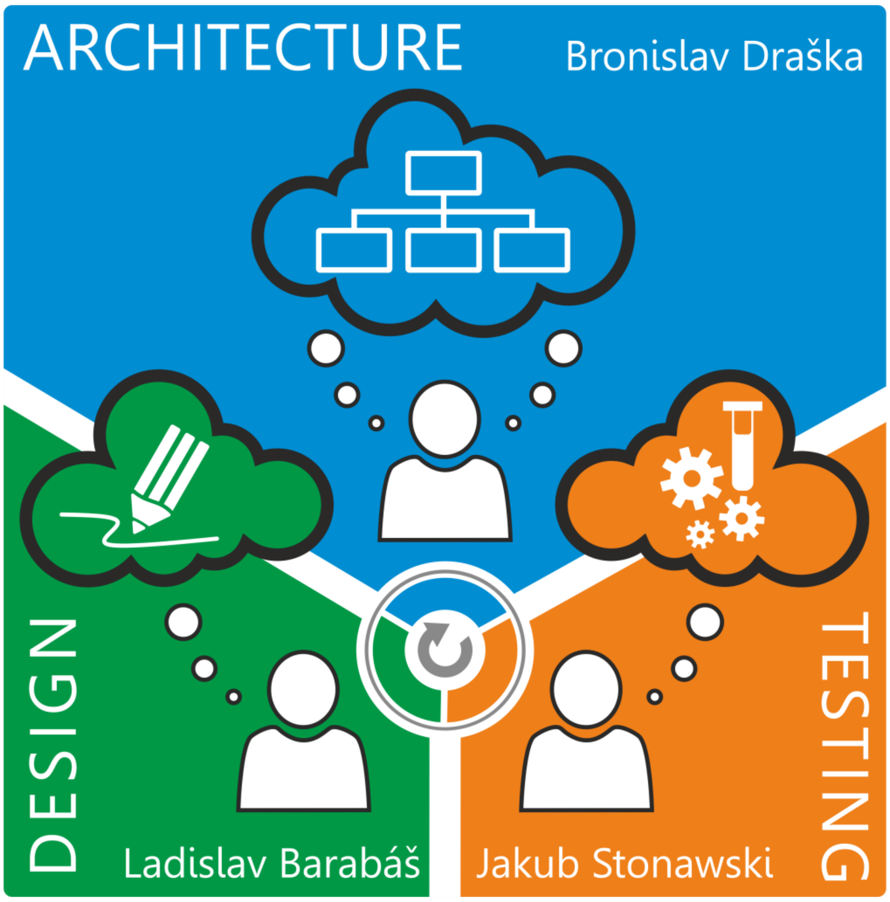


Figure 1 - Team work

Next to implementation related work the team also worked on many organizational and support issues:

- **Jakub Stonawski** participated on communication with customer, presentations of the main ideas, solutions and demo applications.
- **Ladislav Barabáš** took care of reports and notes from all meetings with customer.
- **Bronislav Draška** was responsible for TFS (Team Foundation Server) as well as SQL (Structured Query Language) server. He also managed the work of the whole team and communicated with customer.

There was also amount of work that was participated together, for example design of the whole system, documentation, discussions during the meeting and solving of major problems as well as key functions during the implementation.

3 Tehohydro company overview

Tehohydro represents a small company that provides hydraulics and pneumatics services. This Finnish company was founded in 1983 (European Commision, 2005). Tehohydro operates in industry, which is mainly focused on the production, maintenance and repairs of the hydraulic and pneumatic tool (Tehohydro Oy, n.d.). This company makes a wide range of tests and analysis, which task is checking correct functionality of many products. They perform also measuring techniques for checking levels of impurities in oil and water and all data are collected by special devices. These procedures are very important for solving problems with which the customers come, because mechanics can advise if maintaining or repairing is a better option than purchasing of a new product. They offer repairing and testing of many products like: hydraulic systems, pumps, cylinders, engines, valves, etc. All mechanics are specialized and regularly trained in their field.

Most of their customers are medium-sized and large manufacturing companies and machine shops. The main place of work is South Karelia and Kymenlaakso. The factory is located in Lappeenranta and it serves also like a shop, where customers can buy (or bring to repair) all necessary components and where most of the repairs are made. Figure 2 shows the factory in Lappeenranta.

Figure 2 - Tehohydro company located in Lappeenranta

Surely for this service in such a company it is necessary to have a basic information system, which is able to store all important data from the production and sale to database (Paisley University, 2003). Large part of these data consists of information about customers, employees and their working hours. All these data bring together a structure called job-ticket and the information system is created directly for keeping this structure. Salesmen use the system for storing the information about new customers, mechanics have to keep records about their finished work and they use this system for these purposes.

4 Current system

In the following paragraphs, the current Job-Ticket system is described as it is now used in Tehohydro. There are also the basic characteristics of technology which was used in the current solution. Because it is quite a specific information system, which was created exactly for one company the first subchapter starts with the description of the main functions which job-ticket system should contain.

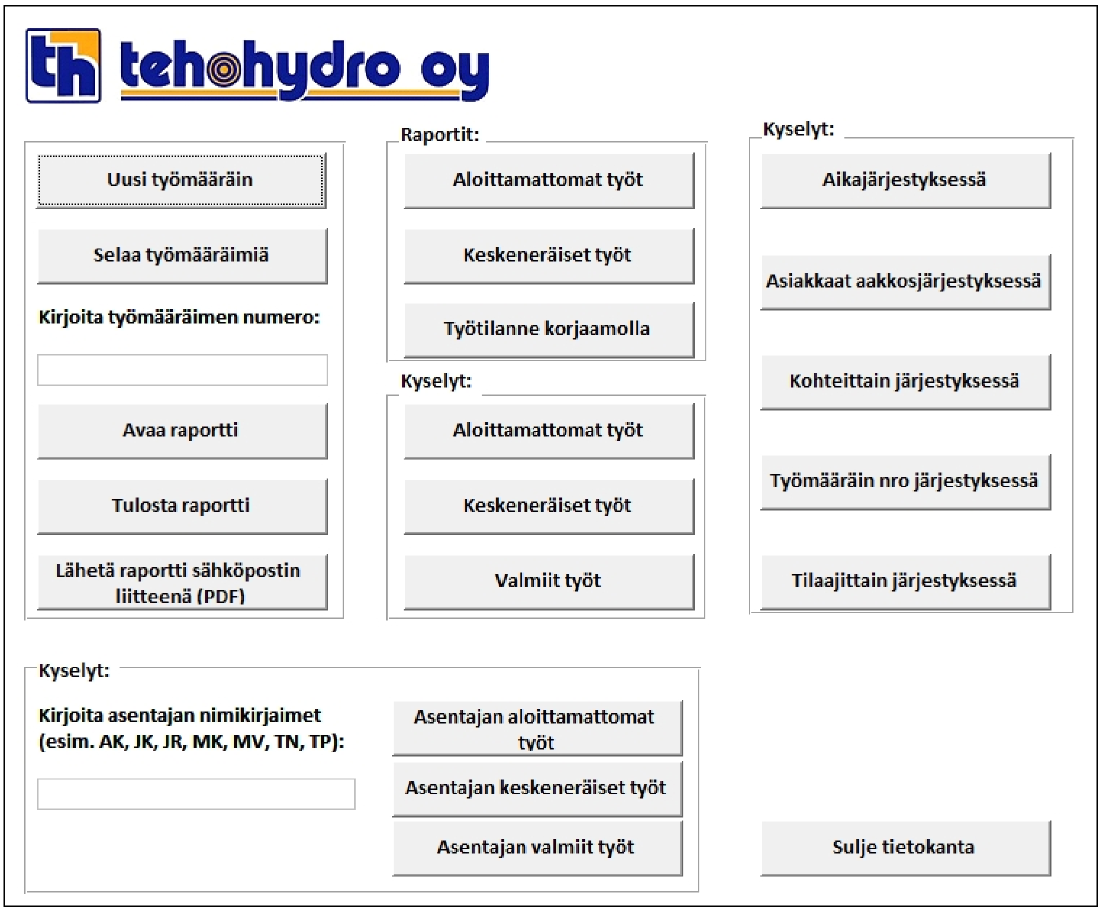

tehohydro oy
Uusi työmääräin
Selaa työmääräimiä
Kirjoita työmääräimen numero:
Avaa raportti
Tulosta raportti
Lähetä raportti sähköpostin liitteenä (PDF)
Raportit:
Aloittamattomat työt
Keskeneräiset työt
Työtilanne korjaamolla
Kyselyt:
Aloittamattomat työt
Keskeneräiset työt
Valmiit työt
Kyselyt:
Aikajärjestyksessä
Asiakkaat aakkosjärjestyksessä
Kohteittain järjestyksessä
Työmääräin nro järjestyksessä
Tilaajittain järjestyksessä
Kyselyt:
Kirjoita asentajan nimikirjaimet (esim. AK, JK, JR, MK, MV, TN, TP):
Asentajan aloittamattomat työt
Asentajan keskeneräiset työt
Asentajan valmiit työt
Sulje tietokanta

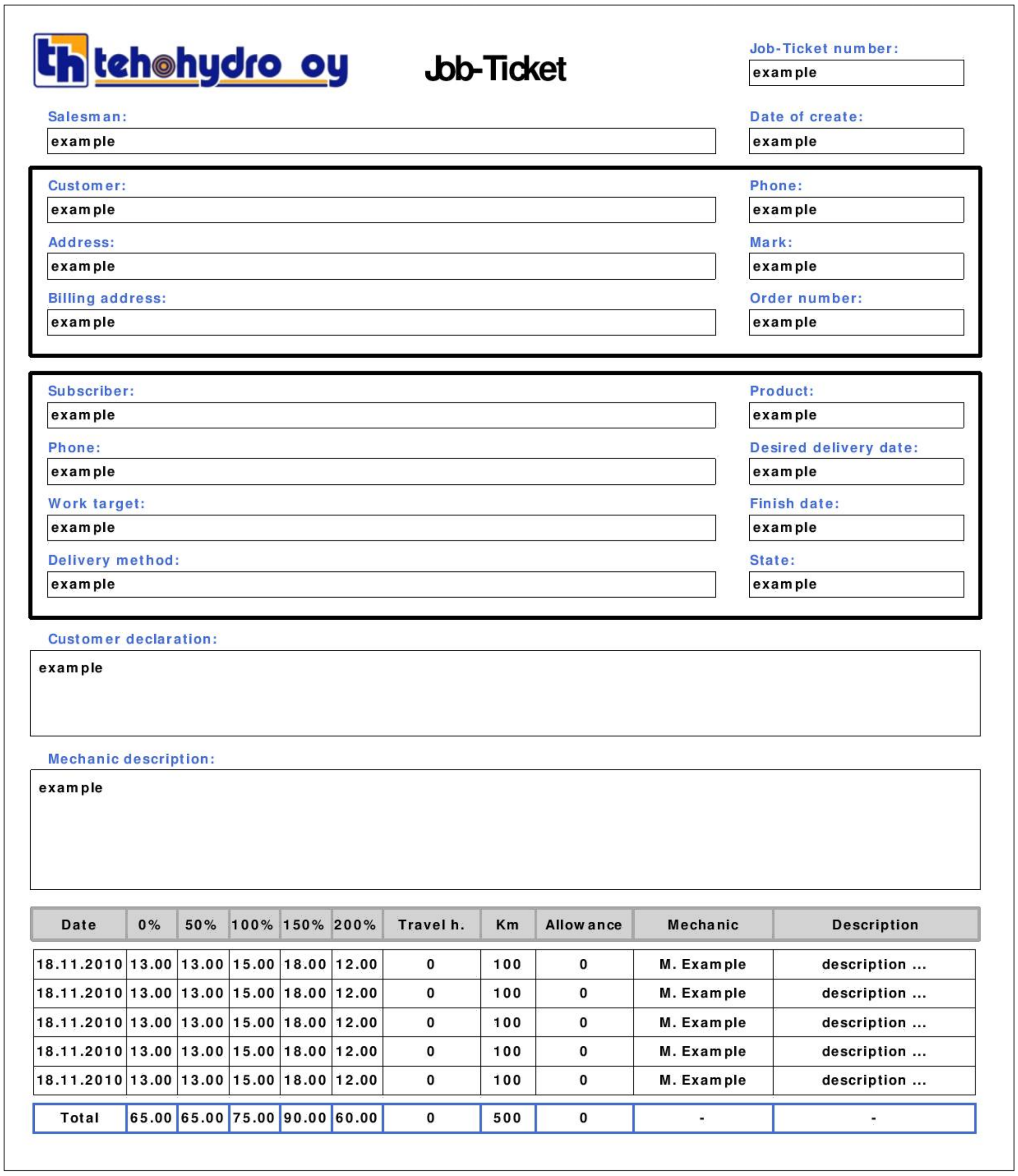
Figure 3 - Current Job-Ticket system

4.1 Job-Ticket system

Because Tehohydro is a company with hundreds of customers, they have to manage all works related to repairs or development of products in a very effective way. Nowadays for this type of a company it is almost impossible to manage all paperwork related to each job by hand. For this propose they are using a specific information system which enables any employee of the company to create a new order of job for a specific customer. Then they can add other important information such as customer's declaration about the problem, estimated time of finish of this order, etc. Finally they save this job-ticket to the database and from this point every mechanic when he will work on this order can add his own working hours. In a very late stage the head of the company can easily check how many mechanics were working on a specific job-ticket, how many orders the company had during a specific period of time and other important information like if there are any job-tickets with delay of finish time.

Every job-ticket should contain:

- customer and all necessary data about him, like address, phone, etc.
- salesman (this person received order from customer, it can be also mechanic himself)
- specification of problem (from both sides, customer's and mechanic's)
- responsible mechanic (primary mechanic)
- list of mechanics which ever worked on this job (with numbers of hours what they spend on it).



tehohydro oy

Job-Ticket

Job-Ticket number:	example
Salesman:	example
Date of create:	example
Customer:	example
Phone:	example
Address:	example
Mark:	example
Billing address:	example
Order number:	example
Subscriber:	example
Product:	example
Phone:	example
Desired delivery date:	example
Work target:	example
Finish date:	example
Delivery method:	example
State:	example
Customer declaration:	example
Mechanic description:	example

Date	0%	50%	100%	150%	200%	Travel h.	Km	Allowance	Mechanic	Description
18.11.2010	13.00	13.00	15.00	18.00	12.00	0	100	0	M. Example	description ...
18.11.2010	13.00	13.00	15.00	18.00	12.00	0	100	0	M. Example	description ...
18.11.2010	13.00	13.00	15.00	18.00	12.00	0	100	0	M. Example	description ...
18.11.2010	13.00	13.00	15.00	18.00	12.00	0	100	0	M. Example	description ...
18.11.2010	13.00	13.00	15.00	18.00	12.00	0	100	0	M. Example	description ...
Total	65.00	65.00	75.00	90.00	60.00	0	500	0	-	-

Figure 4 - Job-ticket paper form example

4.2 Used technology

The whole system is made in MS (Microsoft) Access. Most part of this system is made in design view. Behavior and properties of the system made in this way are based on relationships of tables, their attributes and constrains. This is an advantage for creating this kind of application but every saved time can be easily lost in any future changes. (Blue Claw Database Design, n.d.)

MS Access

This software was developed by Microsoft and it is basically used for management of relational databases. This basic type of database is based on a so-called relational model which means that tables are connected to each other in a specific logical way. This model was developed by Dr. E.F. Codd in 1969. (Codd, 1969)

MS Access also enables the user to create own applications that can be easily deployed on every computer with this software. Creation of this application can be based on mouse-clicking actions during which the user just chooses what the application should do. It is a strong tool for users who have no experiences with programming with the help of real programming languages. MS Access also contains a lot of templates which can even more help the user to create his/her own application. (Microsoft Corporation, n.d.)

There is also a possibility of programming using VBA (Visual Basic for Applications) language. VBA is similar to VB (Visual Basic) and uses VB Runtime Library. This language is used for macros in MS Access.

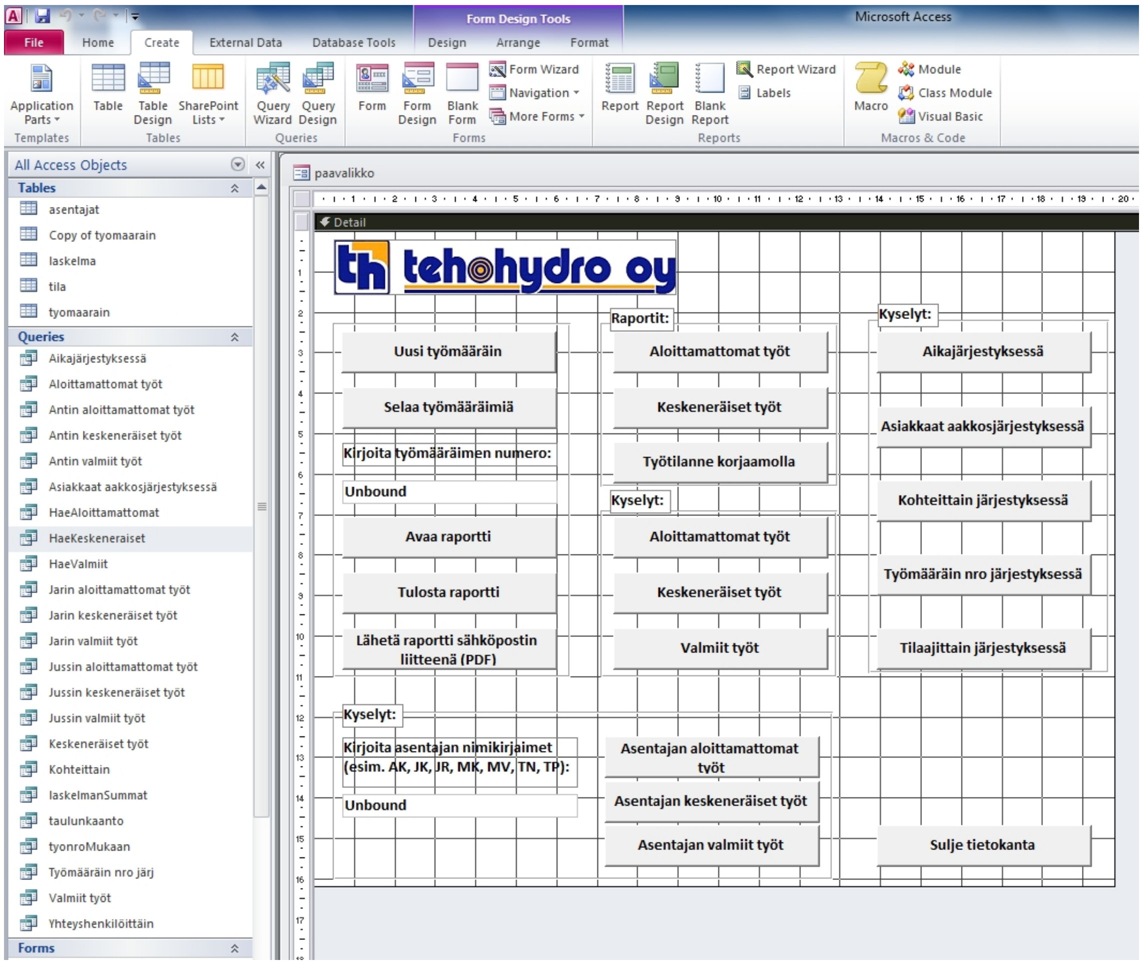


Figure 5 - MS Access environment

4.3 Analysis of current system

During the analysis of the system many issues on the database were discovered:

1) Tables are not even in 1.NF (First Normal Form). For example the whole address is represented by one attribute.
2) Name and surname of mechanics are stored as one attribute and moreover, it is a primary key.
3) Customers and their subscribers are stored in a table with job-tickets themselves. This makes the database inconsistent since there is nothing handling unification of customers.
4) Job-tickets have primary key 'tyomaarainID' which is automatically set by auto incrementation. But another attributed called 'tyomaarainNro' is

used in the whole system as primary key. Additionally ‘tyomaarainNro’ do not have to be unique, can be edited by user and it is not even required.

The entity-relationships of the database used in this system are shown in Figure 6.

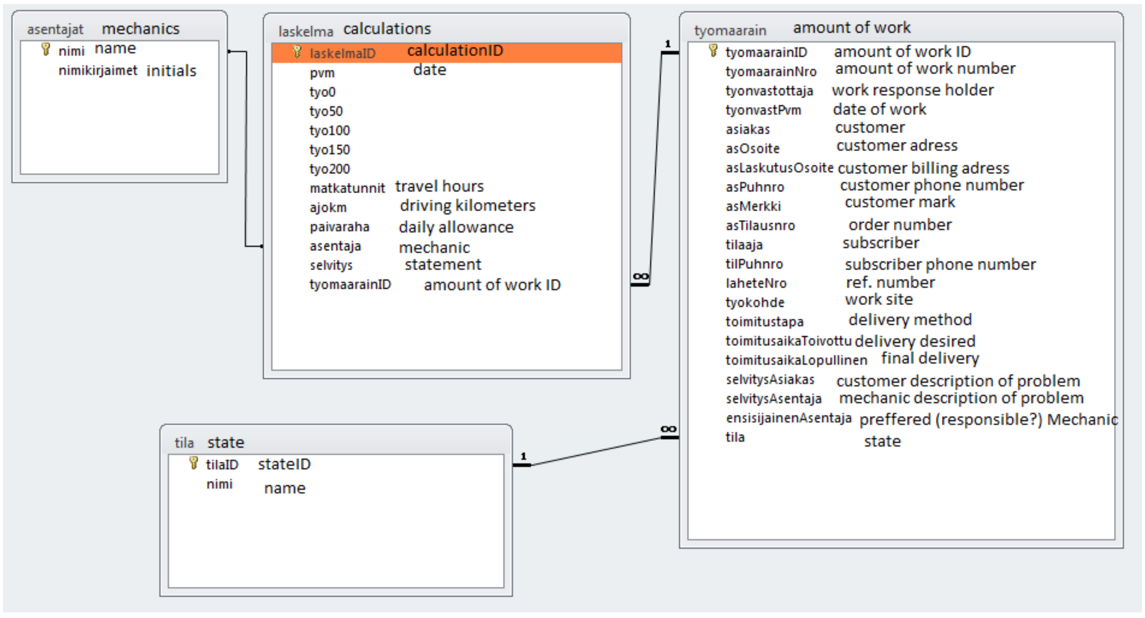

asentajat mechanics
nimi name
nimikirjaimet initials
laskelma calculations
laskelmaID calculationID
pvm date
tyo0
tyo50
tyo100
tyo150
tyo200
matkatunnit travel hours
ajokm driving kilometers
paivaraha daily allowance
asentaja mechanic
selvitys statement
tyomaarainID amount of work ID
tyomaarain amount of work
tyomaarainID amount of work ID
tyomaarainNro amount of work number
tyonvastottaja work response holder
tyonvastPvm date of work
asiakas customer
asOsoite customer adress
asLaskutusOsoite customer billing adress
asPuhnro customer phone number
asMerkki customer mark
asTilausnro order number
tilaaja subscriber
tilPuhnro subscriber phone number
laheteNro ref. number
tyokohde work site
toimitustapa delivery method
toimitusaikaToivottu delivery desired
toimitusaikaLopullinen final delivery
selvitysAsiakas customer description of problem
selvitysAsentaja mechanic description of problem
ensisijainenAsentaja preffered (responsible?) Mechanic
tila state
tila state
tilaID stateID
nimi name
1
∞
1
∞

Figure 6 - Entity-relationship diagram of the current system

The final version of this improvement would probably be unstable and also lot of functions which were requested by the customer could not be created due to MS Access environment limitations.

5 New system

Based on the previous facts it was really necessary to create a new system from scratch which also means that more optimal technologies and approaches can be used. The scope of this project is to develop a windows-form application that will provide all described functions of Job-Ticket system. There is also need to develop a database for this system. The architecture of this system will be based on Domain Model design pattern. (Lasater, 2010) That means that no server application is needed and all communication between the database and the application instances will be based on transactions and their isolation levels. MS SQL Server is a strong enough tool to handle all of this. Considering the scope of the whole system it was also decided to create own ORM (Object-relational mapping) for communication between the windows-form application and the database.

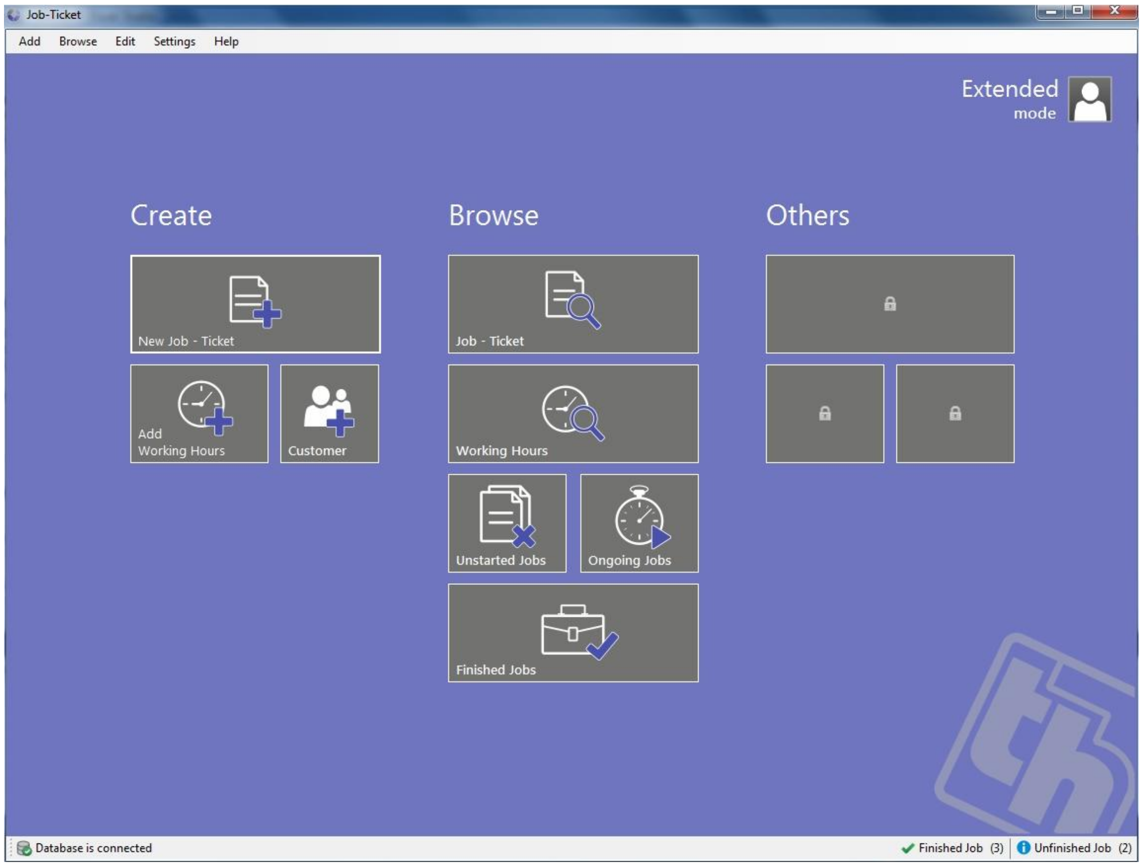


Figure 7 - New Job-Ticket system

5.1 Used technology

Based on discussions with the customer it was decided to develop the system as a standard Win-Form application. For developing the system C# programming language and MS .NET Framework 4.0 were used. This choice was based on previous experiences of all members of the team. On database level standard SQL queries and commands for manipulation with the data were used. Some commands were supplemented with short anonymous procedures in T-SQL (Transact-SQL) language. Own ORM was also used instead of any frameworks.

5.2 Used development tools

During the development of the new system many tools were used. These tools were used in the implementation phase, for presenting the results to the customer or for scheduling the work in a team. Among them belong MS Visual studio 2010, Team Foundation Server, OneNote, Prezi and Gantt Project.

5.2.1 MS Visual Studio 2010

As a main tool to create the new Job-Ticket system MS Visual studio 2010 in two different versions was chosen. The first version, Professional, was used for most of functionality in the system. The Ultimate version was necessary for final testing and optimization of the whole system because it contains a special tool for these propose. It is able to analyze the application during run-time and after that it shows where a code is the slowest. (Microsoft Corporation, n.d.) After this analysis it is possible to optimize the code manually.

MS Visual studio is an ideal development environment because it provides very useful tools for developing these types of information systems. Additionally this tool has Team Foundation Server (5.2.2).

MS Visual studio also provides an environment where the GUI (Graphical User Interface) was created. The Figure 8 shows, how the environment looks.

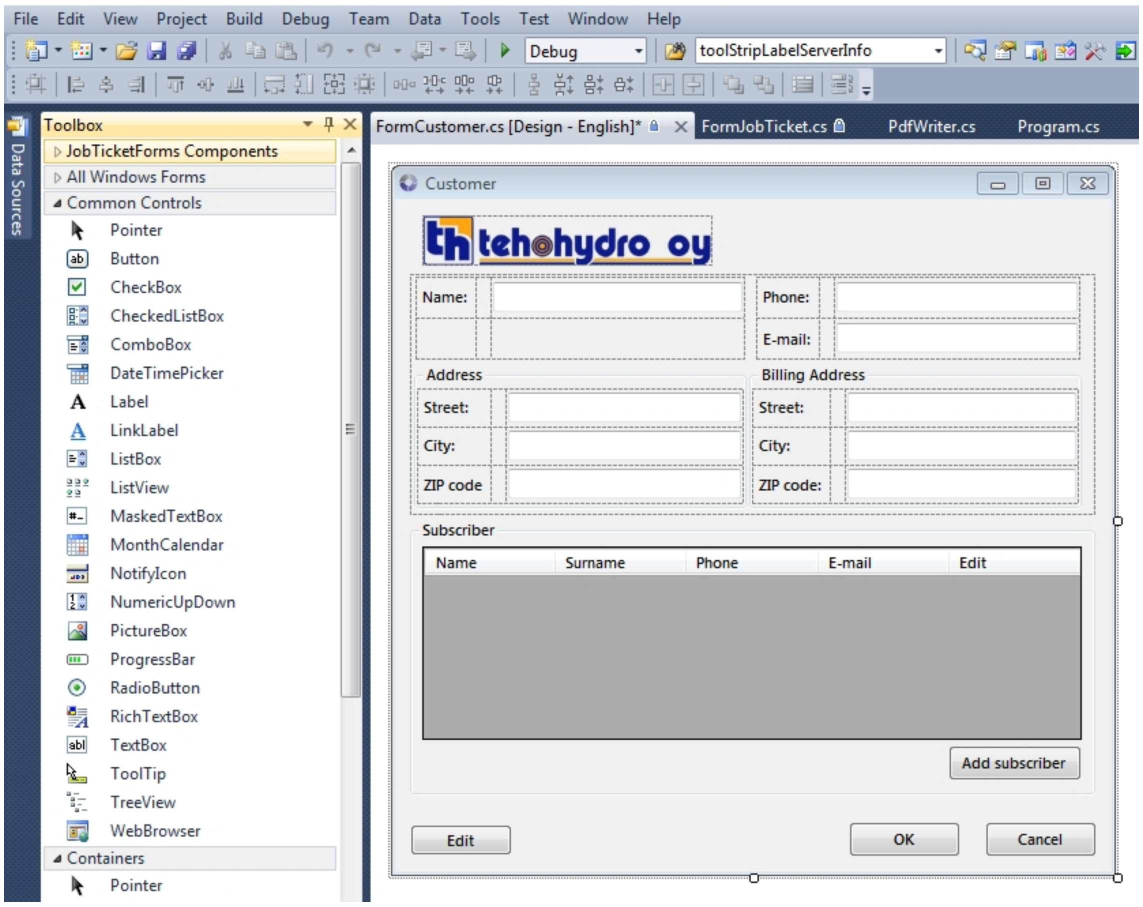
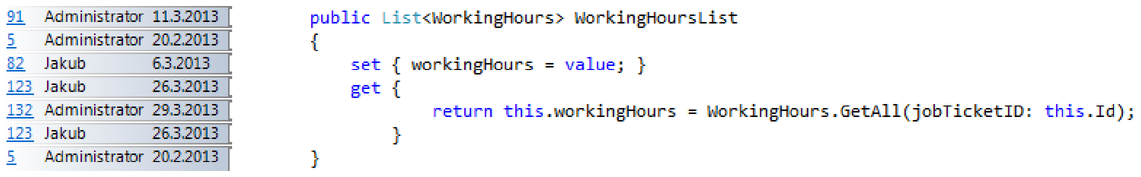


Figure 8 - MS Visual Studio environment

5.2.2 Team Foundation Server

TFS (Team Foundation Server) supports agile development practices, multiple IDE (Integrated Development Environment) and platforms locally or in the cloud and gives the tools you need to effectively manage software development projects throughout the IT lifecycle. (Microsoft Corporation, n.d.)

TFS was primarily a tool for merging the code of multiple users working on the same project. It is also a useful tool for backup and archiving of historic versions where the name of developer as author of the last change for each row can be seen. Whole source code history with names of authors is also provided as Figure 9 shows.

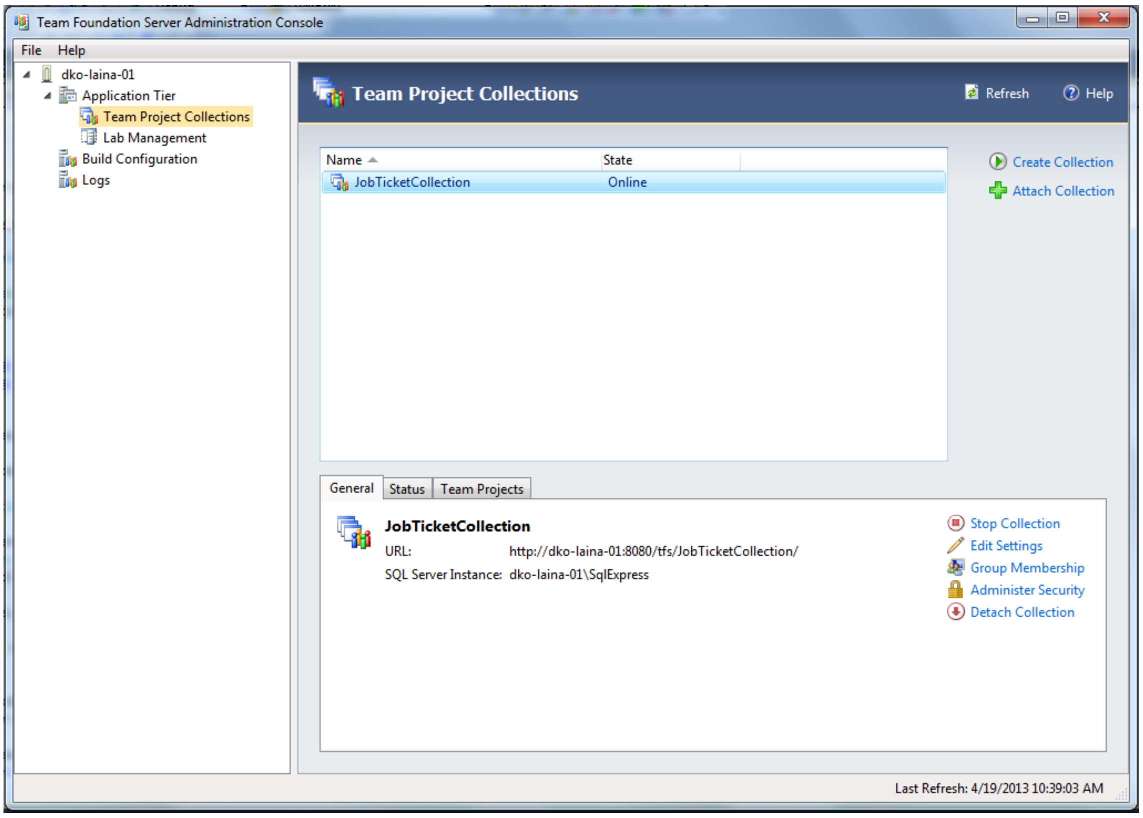


91	Administrator	11.3.2013
5	Administrator	20.2.2013
82	Jakub	6.3.2013
123	Jakub	26.3.2013
132	Administrator	29.3.2013
123	Jakub	26.3.2013
5	Administrator	20.2.2013

```
public List<WorkingHours> WorkingHoursList
{
    set { workingHours = value; }
    get {
            return this.workingHours = WorkingHours.GetAll(jobTicketID: this.Id);
        }
}
```

Figure 9 - MS Visual Studio Source code history

Management of TFS server is done by Team Foundation Server Administration Console shown in Figure 10. Source control is managed via Source control Explorer in Visual Studio.



Name	State
JobTicketCollection	Online

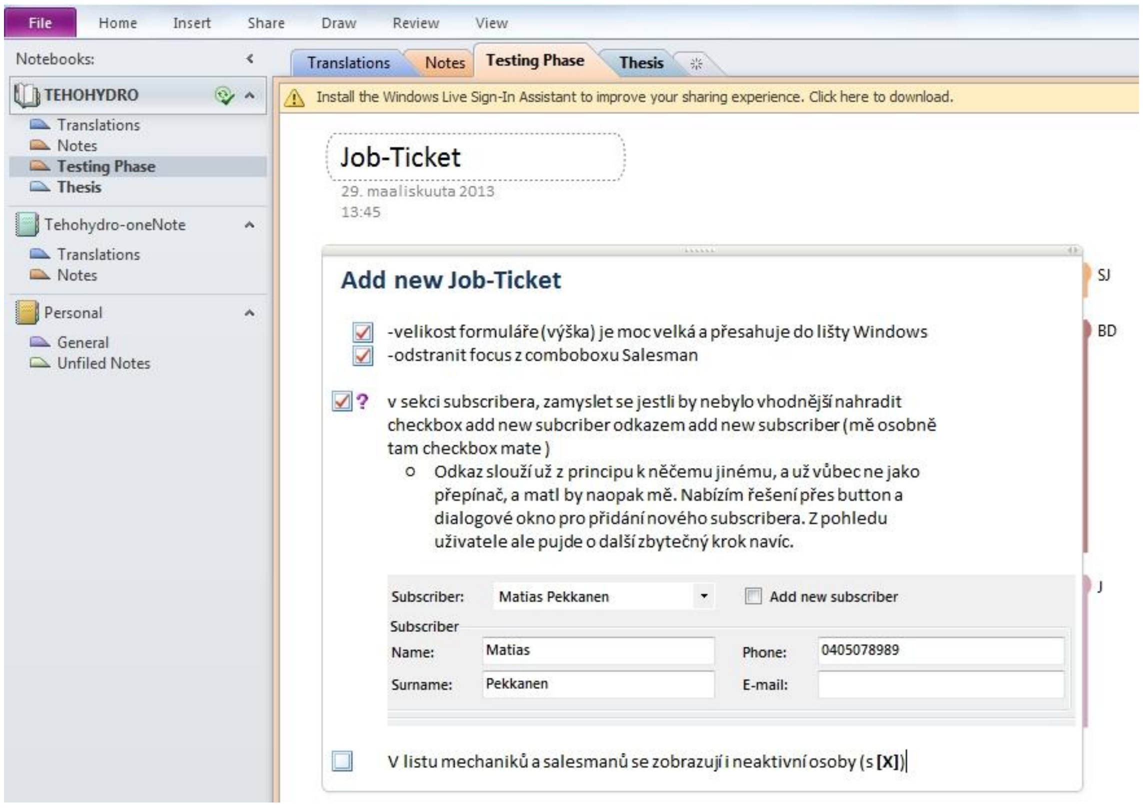
JobTicketCollection
URL: http://dko-laina-01:8080/tfs/JobTicketCollection/
SQL Server Instance: dko-laina-01\SqlExpress

Last Refresh: 4/19/2013 10:39:03 AM

Figure 10 - Team Foundation Server Administration Console

5.2.3 OneNote

OneNote is an improved notepad-style application made by Microsoft which enables to make and share all notes and to-do lists inside the team. (Microsoft Corporation, n.d.) It can be part of MS Office pack but is also available separately. For this project OneNote was used for sharing notes and for distribution of work between all developers. One of the most used parts of this application was Snapshot tool which enables to crop the taken screenshot and save it to clipboard. In shared notebooks the identification of the author of the last change is shown as it can be seen in Figure 11.



Install the Windows Live Sign-In Assistant to improve your sharing experience. Click here to download.

Job-Ticket
29. maaliskuuta 2013
13:45

Add new Job-Ticket

- [x] -velikost formuláře (výška) je moc velká a přesahuje do lišty Windows
- [x] -odstranit focus z comboboxu Salesman

- [x] ? v sekci subscribera, zamyslet se jestli by nebylo vhodnější nahradit checkbox add new subcriber odkazem add new subscriber (mě osobně tam checkbox mate)
    - Odkaz slouží už z principu k něčemu jinému, a už vůbec ne jako přepínač, a matl by naopak mě. Nabízím řešení přes button a dialogové okno pro přidání nového subscribera. Z pohledu uživatele ale pujde o další zbytečný krok navíc.

Subscriber: Matias Pekkanen ☐ Add new subscriber

Subscriber

Name:	Matias	Phone:	0405078989
Surname:	Pekkanen	E-mail:	

- [ ] V listu mechaniků a salesmanů se zobrazují i neaktivní osoby (s **[X]**)

Figure 11 - OneNote environment

Synchronization can be invoked manually by the user but in default it is made automatically when changes are finished. Notebooks in OneNote can be shared via SharePoint Server or cloud service SkyDrive. For SkyDrive synchronization Microsoft Live account is used.

5.2.4 Prezi

The presentation of the proper idea about product to customer is one of the most important parts of the process of the product offer. It can be done successful and the customer is satisfied. This means that everything is clearly explained and every important point is mentioned as easily as possible. But on the other hand, these important parts of concept development are not always met and then misunderstanding or confusion from the customer’s side could come. To avoid these potential threats, the team decided to use Prezi as a tool to create the bridge of imagination between the two sides – the team and the customer.

Prezi is a very simple cloud-based presentation software, which offers easily to users make own presentation via internet browser. (Prezi Inc., n.d.) This service is freeware and the only thing, what is needed is to be registered as Prezi user. It is mostly widespread as online service, so the internet connection is required for using. But there is also a possibility to use Prezi offline by using the desktop application. This desktop application is an extended version, which is chargeable. Freeware version was fully sufficient for the project purposes.

Working in Prezi environment is intuitive and easy. The user will quickly learn the main functions. It is possible to use many predefined templates, which are offered for creating new presentation before. There is also a possibility to convert a Microsoft PowerPoint presentation directly on Prezi canvas. All created presentations are automatically saved in user's profile, but presentation can also be exported to a flash file and saved in local computer. Figure 12 shows, how Prezi environment looks.

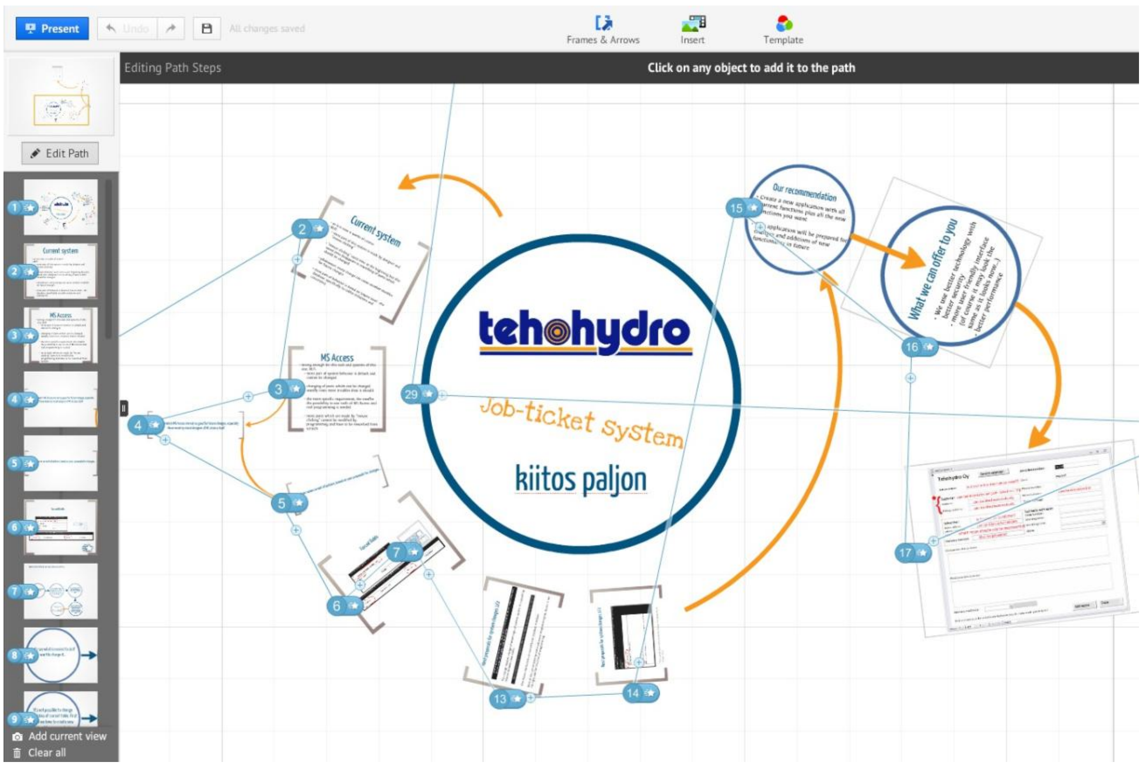


Figure 12 - Prezi environment

The main difference compared to usual presentations (for example Microsoft PowerPoint) is that the user works only with one canvas instead of traditional slides. This canvas contains all objects, just like text, images, videos, sounds, etc., and these objects are in a particular order zoomed in and out depending on the user's choice. As a result an easily understandable presentation can be created, which does not bore the audience and what more – can impress them. The main reason for choosing this tool was because of its simplicity, ability to engage the viewers and also because Prezi is about creativity.

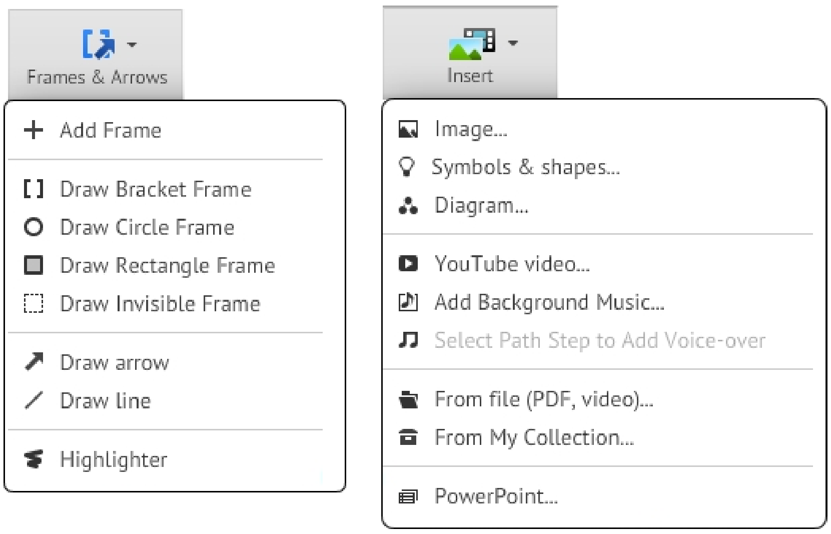
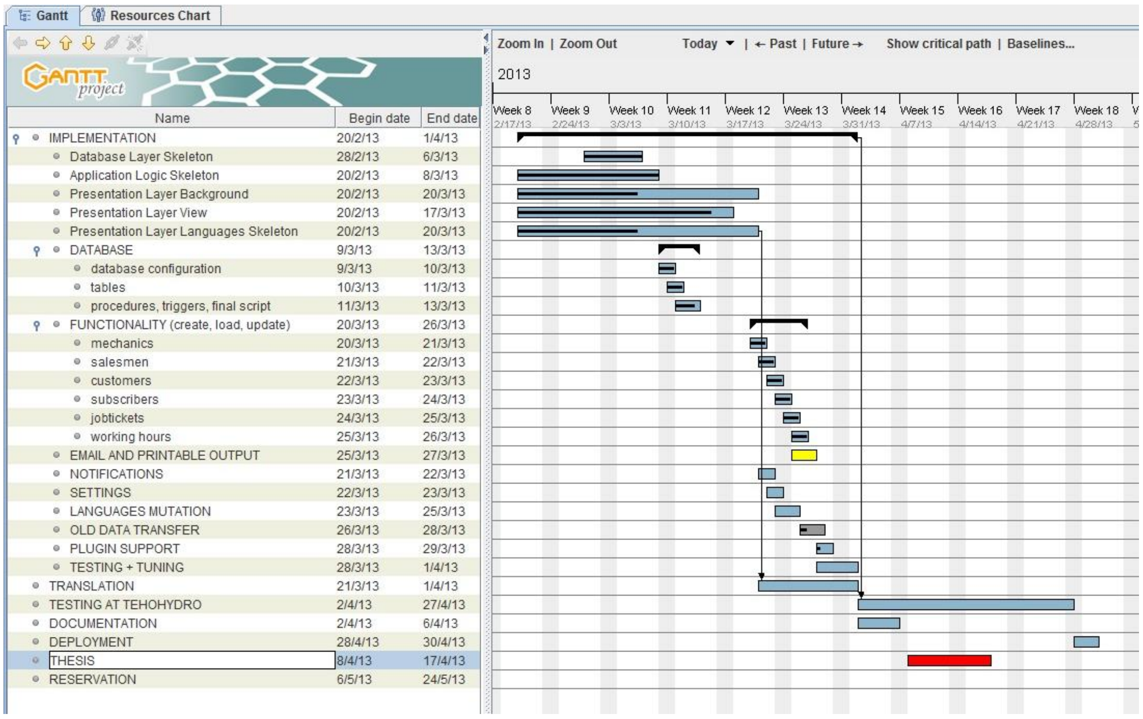


Figure 13 - Prezi's main functions

5.2.5 GanttProject

If any team starts to develop a specific project (for example information system), the work inside the team must be organized if good results are expected. Each member should know his/her own position in a team. The team should also decide together on the project schedule, which will divide the whole project work into single phases and tasks. This is the first step leading to successful objectives of the work. The team has decided to use GanttProject for these purposes. (Barashev & Thomas, n.d.)

GanttProject is a work scheduling and time management desktop application. This software is freeware and its code is open source. It is programmed in Java language so it means that software is cross-platform. The user can create project tasks, for which the start and finish date are set, it is easy to watch progress through the whole development. The resource load chart allows writing records about the workload of each member of the team. The schedule plan with individual tasks and their progress is possible export to PDF document. The Figure 14 shows, how the environment looks.



Name	Begin date	End date
IMPLEMENTATION	20/2/13	1/4/13
Database Layer Skeleton	28/2/13	6/3/13
Application Logic Skeleton	20/2/13	8/3/13
Presentation Layer Background	20/2/13	20/3/13
Presentation Layer View	20/2/13	17/3/13
Presentation Layer Languages Skeleton	20/2/13	20/3/13
DATABASE	9/3/13	13/3/13
database configuration	9/3/13	10/3/13
tables	10/3/13	11/3/13
procedures, triggers, final script	11/3/13	13/3/13
FUNCTIONALITY (create, load, update)	20/3/13	26/3/13
mechanics	20/3/13	21/3/13
salesmen	21/3/13	22/3/13
customers	22/3/13	23/3/13
subscribers	23/3/13	24/3/13
jobtickets	24/3/13	25/3/13
working hours	25/3/13	26/3/13
EMAIL AND PRINTABLE OUTPUT	25/3/13	27/3/13
NOTIFICATIONS	21/3/13	22/3/13
SETTINGS	22/3/13	23/3/13
LANGUAGES MUTATION	23/3/13	25/3/13
OLD DATA TRANSFER	26/3/13	28/3/13
PLUGIN SUPPORT	28/3/13	29/3/13
TESTING + TUNING	28/3/13	1/4/13
TRANSLATION	21/3/13	1/4/13
TESTING AT TEHOHYDRO	2/4/13	27/4/13
DOCUMENTATION	2/4/13	6/4/13
DEPLOYMENT	28/4/13	30/4/13
THESIS	8/4/13	17/4/13
RESERVATION	6/5/13	24/5/13

Figure 14 - GanttProject environment

5.3 Architecture

The architecture of the application is based on Domain Model DP (design pattern). This enables to divide application into three layers:

- **Presentation layer**

  This layer takes care of the user interface and its behavior. From this layer domain logic layer is controlled.

- **Domain logic layer**

  All main logic parts of the application according to the system functionality requirements are placed in this layer. Greater part of the application behavior is implemented in this layer. This layer is dependent on the database layer through which it obtains the data for its work.

- **Database layer**

  Database layer provides the data for layers above. This layer also enables to manipulate with the data as well. Data itself are placed in data storage outside the application, like MS SQL Server database in this case. Database layer is thus dependent on the type of data storage.

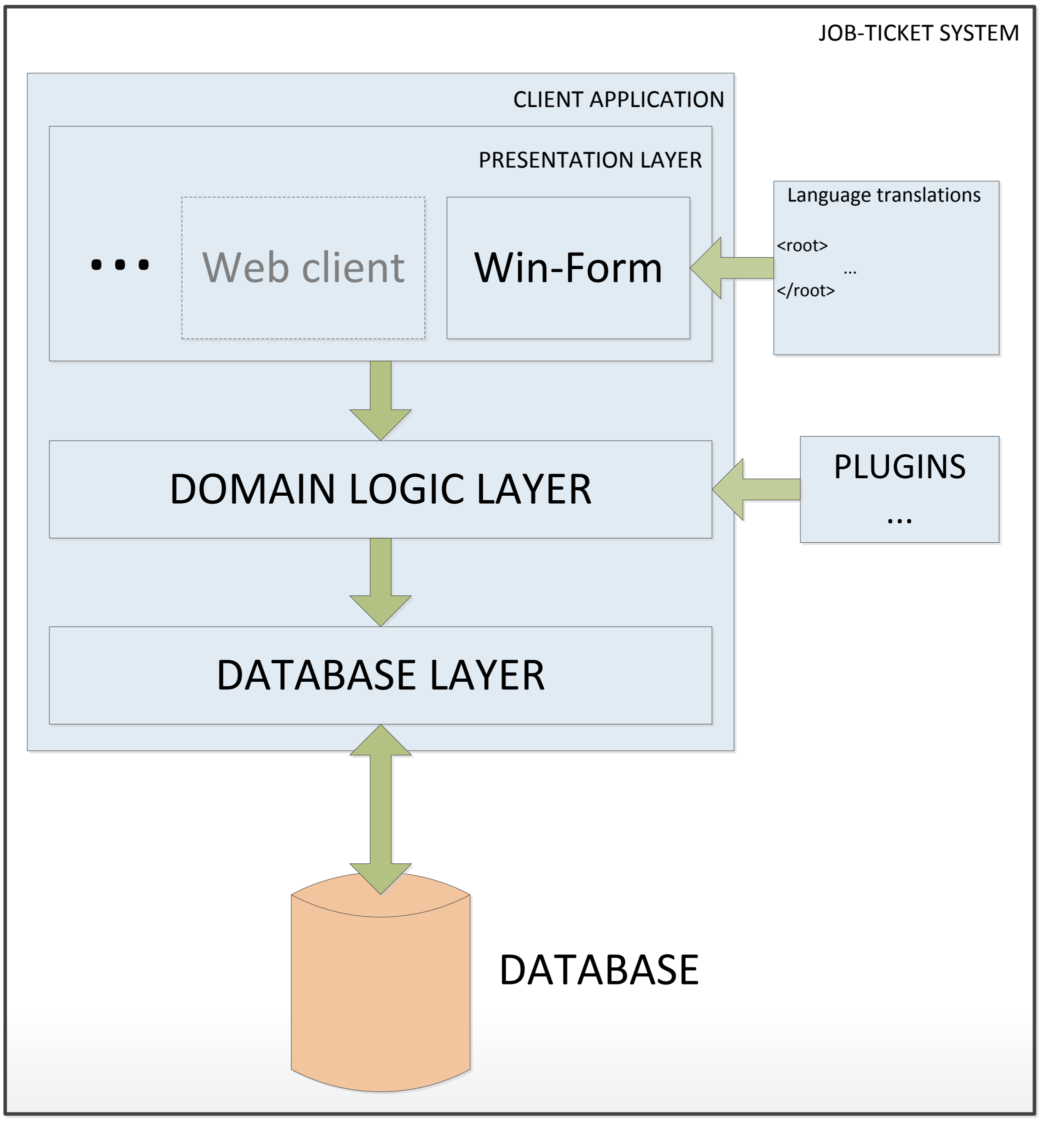


Figure 15 - System architecture

There is no dependency of any layer on the layer above. Due to that fact implementation and maintenance of the source code of the application is clearer. Another reason why this pattern was chosen is the possibility of replacing the presentation layer with any other type of user interface of .NET technology without touching the application logic or even creating a new one.

According to customer’s requirements the application was supplemented by support of plug-in and Multilanguage user interface. Both of them are described in the chapter Other Features.

5.4 Database

This chapter describes everything important about the database used in Job-Ticket system.

5.4.1 E-R Diagram

All entities in the database are shown in Figure 16, all relations between them as well as types of these relations. Primary and foreign keys are shown there too.

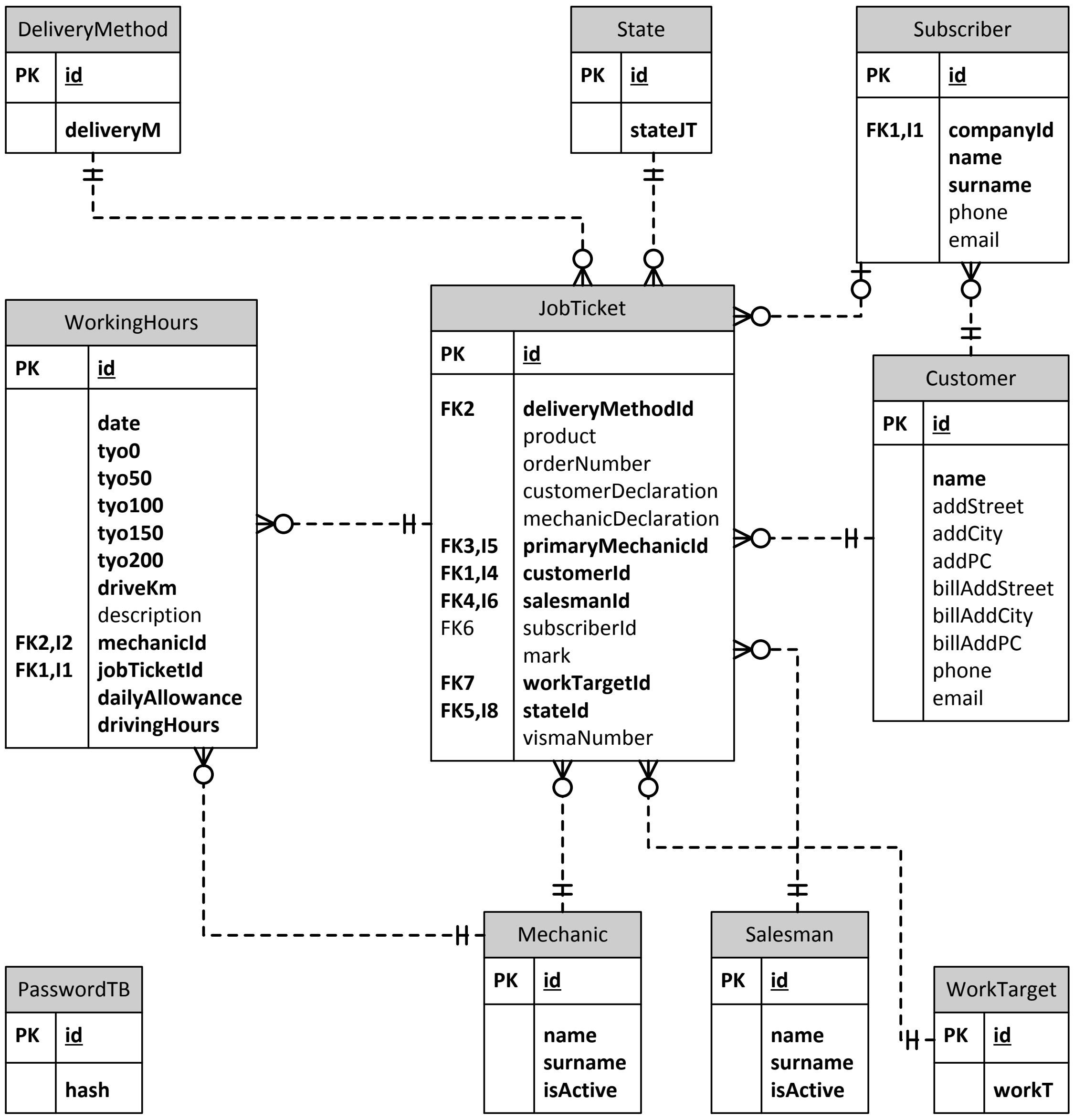


Figure 16 - E-R Diagram of a new system

5.4.2 Database specifications

As a primary key auto incrementing number of integer type is used. This enables to store up to $2^{31} - 1$ records in each table since it is the maximum value of integer and the first inserted has value '1'. All primary keys are solved in the same way for all entities.

For descriptions 'nvarchar' types with unlimited number of character are used. The maximum length of 'nvarchar' stored in one row is 4000 characters. When this length is exceeded, the data in specific row is replaced by pointer to one or more pages where these data really are stored.

There was also need to secure best performance for filters. For this purpose indexes for important attributes were created. For primary keys indexes are created automatically by DBMS (Database Management System). Indexes reduce performance for updating since the index tree must be reorganized. On the other hand they increase performance for queries because records do not have to be read sequentially. Job-ticket system does not expect many parallel updates but queries of filters have great demands on DBMS. Nevertheless MS SQL Server is a strong enough system for the amount of data flow that Job-Ticket system can produce and the user will not recognize any reduction of performance during updating the database.

5.4.3 Old Data Transferring

Before starting an analysis, the team agreed with the customer that data from the old system will be transferred to the new one. Unfortunately information about the real condition of data in the old system was not quite clear. Relations between entities were analyzed and from that point of view it looked like there should not be any crucial problem. But after taking a closer look at records in tables the team discovered a lot of empty domains as well as domains with incorrect values for important attributes which were needed for transferring the data. The database needed crucial improvements because of its inconsistent and lack of integrity.

Two ways were found to solve those problems. The first one was to start from the scratch and not transfer any old data to the new system. The second solution was to transfer only the correct data and then try to find out which records can be considered as correct ones. After explaining the problems to the customer the team decided not to transfer the old data at all. This decision saved some time but on the other hand a phase for testing the system with data from the old system was also included in the plan. For that purpose only a part of data was transferred from the old system. If there were records with incorrect values the values were replaced with any random but correct values. The amount of the data was huge enough for debugging as well as optimization of speed of the new system.

5.5 Functionality

The following text describes the main functions that the new Job-Ticket system contains. All these functions were subject of discussions between the team and Tehohydro representatives as well as other critical behavior of the whole application.

For better clarity this description is divided into subsections that are logically structured according to objects which relate to a specific functionality. Figure 17 shows Use-case diagram for the Job-Ticket system.

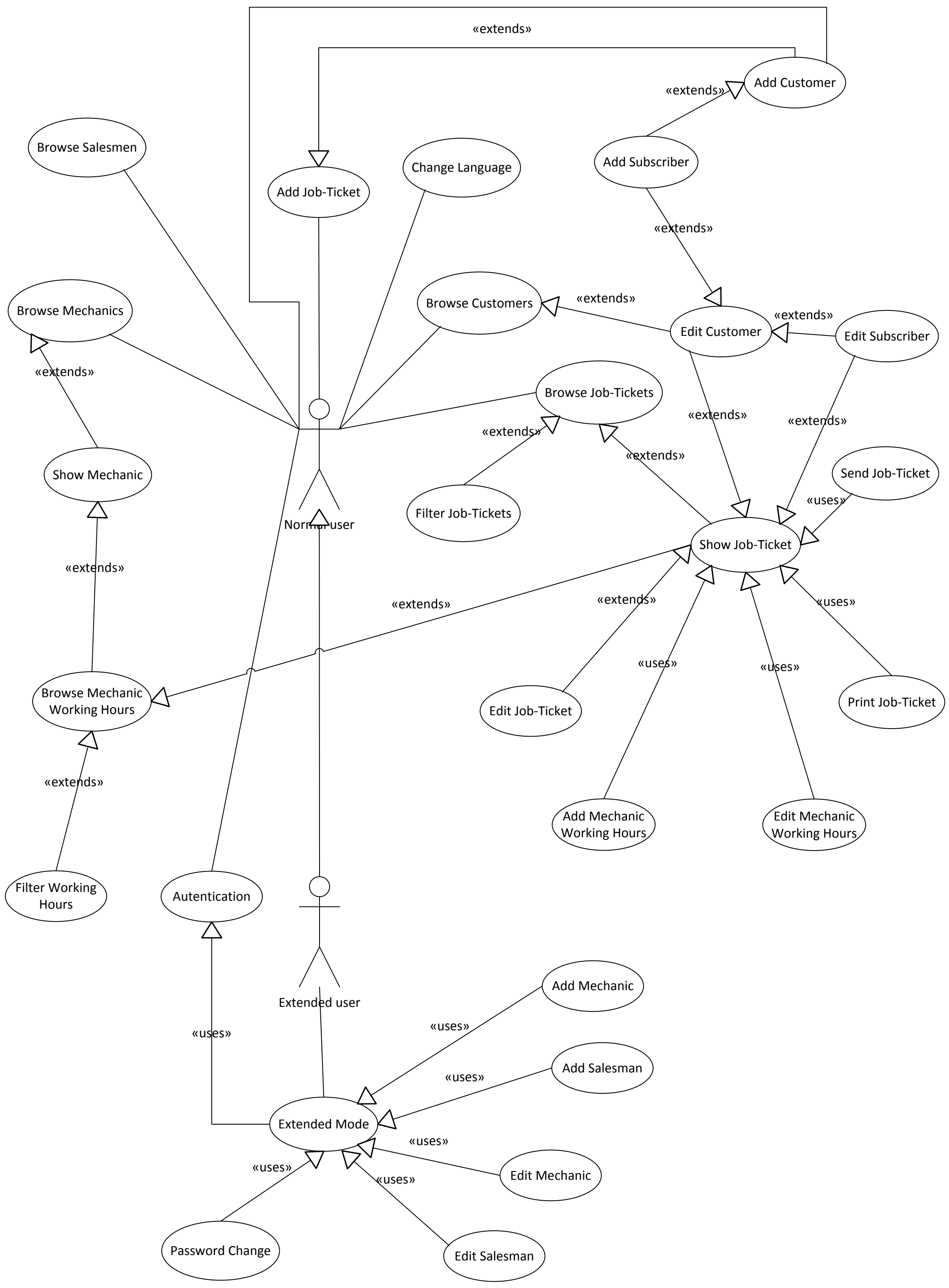


Figure 17 - Use-case diagram for a new system

5.5.1 Customer data processing

Since customer is one of the most important objects in the system everybody who will use Job-Ticket system should be able to add a new customer, edit any data of an already saved customer as well as browse all customers who are saved in the database.

There is also a special type of customer (mostly bigger companies) who has one or even more subscribers. It means that when an employee of Tehohydro wants to create a new job-ticket for this customer he has to fill in also data about a specific subscriber who has the right to act as customer. It means that everybody from Tehohydro should also be able to add a new subscriber or edit any of his data. There must also be a possibility to select a specific subscriber as non-active, which means that from this point there is no possibility to create a new job-ticket with this subscriber.

In a phase of adding a new customer, the user has to fill in all data about the customer which is necessary for saving this customer to the database (name, address ...). If there is any important data missing the customer must not be saved. There must not be a possibility to save a customer, who is already in the database. All mentioned functions for adding a customer also apply for subscriber object. Especially in a subscriber case there must be solved situation that subscriber must have own customer, so there must not be a situation during which the user can create a subscriber without a customer. In the system there are two possibilities how a new customer or a subscriber can be added. The first one is directly from special windows form for this purpose. The second variant will be adding these objects from a form for creation of the new job-ticket.

If somebody would like to edit any data of the customer there are two possibilities how to do this. The first option is that the user chooses a specific person from a list of all customers and then he is able to edit any field. A situation must also be handled when the user deletes any of the required field - these changes must not be saved to the database. The second way how to edit a customer is directly from Job-ticket form, when the user chooses a specific customer, all other fields related to the customer are automatically pre-filled and that user can change any of them. Similar situation is during editing of a subscriber. There are also two options how to do that. The only change is in the first situation, when the user has to select a specific customer from the list of all customers and then select a subscriber who would like to edit.

During the phase of browsing all customers there must also be a function to search across the name of customers which helps the user to find the right object.

5.5.2 Salesman data processing

Salesman is a special type of employee. It is a person who receives an order for the new job-ticket from the customer. It can be (and usually in Tehohydro it is) mechanic but the salesman has to be filled independently of the primary mechanic in Job-ticket form.

Unlike in a case of a customer or subscriber object any user can not add a new salesman. Since the salesman is an official employee of Tehohydro only the extended user can add him to the database.

For an extended user there is also a function for editing the name and surname of the existing salesman. In order to maintain consistency of database, there is no function for deleting the salesman, but the extended user can mark a specific salesman as non-active as in a case of subscriber. As in customer and subscriber case, also here must be handled the situation that the user does not fill in (or delete) the required field.

The only function about the salesman that can be accessed by any user will be browsing all salesmen from the database. There will also be a possibility to search during this phase. Because salesman contains only information about his name and surname there is no need to show any detail information about this object.

5.5.3 Mechanic data processing

Mechanic is the second type of the previously mentioned employee. It means that also for creating and editing this object, the user has to log into the extended mode. After that authentication, the user is able to add a new mechanic, edit any data of a particular mechanic and also select a mechanic as non-active, so there is no possibility to assign this mechanic to new job-ticket or create new working hours for him. As in the previous objects also there protection must be done for saving incorrect records to the database.

For this object, the only one function that can be done, without extended user permission, is to browse all mechanics in the database. Also here there will be a search function for better user experience.

Each mechanic has also his own list of working hours for each job-ticket where he ever worked. It also means a set of special functions for this propose. Each mechanic must be able to add own working hours to a particular job-ticket. This record of working hours must contain special fields for hours of work which have been done during a specific period of working time (tyo0, tyo50 ...). There also must be a possibility to add driving distance to working place, drive hours, daily allowance as well as description of work and date of work. The fact must be solved, that a mechanic cannot add own working hours for a job-ticket which is already marked as finished.

In the list of mechanic's all working hours there must also be a function for filtering working hours which helps the user to find the right record. It should be able to filter by date or period of time as well as by particular job-ticket.

The special function above all mechanics in the database is a statistics tool. It enables the user to check how many hours are spent by a particular mechanic or a couple of mechanics on a specific job-ticket or what is the average performance of all mechanics.

5.5.4 Job-ticket data processing

One of the crucial functions of the whole Job-Ticket system is creating and editing job-ticket. Everybody has to be able to access to any particular job-ticket, viewing all details about it and also modify fields.

During the addition phase, the user has to fill in all necessary information which is required for a successful saving of job-ticket. If this job-ticket is for the customer, who is already in the database, there must be a function which enables the user to choose a specific customer from the list. Then all other already saved data are pre-filled in Job-ticket form. If a customer has any subscribers, a list of these objects must be loaded in Job-ticket form too. If the user chooses one of the subscribers he must also be able to modify any information about this particular subscriber. The user must also be able from this point to create a new one.

During browsing of all customers from Job-ticket form there should also be an option for easy search throw of all records for better user experience.

Also assignment of a primary mechanic and salesman has to be done with the help of lists of all mechanics/salesmen. These lists will only contain mechanics and salesmen who are marked as an active in the database. When all necessary data is filled in Job-ticket form, it can be saved to the database. During this phase there will automatically be filled data about the creation date of this job-ticket.

Because there is expectation that the system contains a really big amount of job-tickets, sophisticated filtering for searching the right job-ticket(s) must be done. There must be a possibility to filter job-tickets by date of creation, date of start work, and date of finished work. The user must also be able to filter by primary mechanics, customers, and salesman as well as by the current state of job-ticket.

When a user will open any job-ticket for editing he must be able to edit some of the data but not all of them. For example the user must not be able to modify the date of start work on job-ticket which is already in the started phase.

Another important function involves saving of job-ticket to PDF file, printing job-ticket or sending job-ticket via e-mail. After consulting this topic with Tehohydro the team has decided to make these functions together. It means that after the user will want to print or send job-ticket both of actions lead to creation of a PDF file. From this point it is much easier to choose if the user would like to print it now or send it.

5.5.5 Extended mode

As it was mentioned before, some of the functions require a special extended mode. It is a mode only for the head of the company and the main secretary. To enter to this mode the user has to know the password. User interface in this special mode should be a little bit different for better clarity, for example with the help of different colors or other features.

In extended mode the user is able to access a special function mentioned before, like adding a new mechanic or salesman, as well as modify this object. There are also special statistics for the head of the company related to all job-tickets in the database. The user in extended mode is also able to change the password for this mode. Saving of the password has to be done in a safe mode; it must not be saved as plain text in the database.

5.6 Solution of the Job-Ticket system

The following subchapters briefly describe the chosen solutions of all functionalities according to the previous chapter. Problematic and critical parts will be described more deeply. The design patterns will be mentioned as well if any of them are used.

Customers, subscribers, salesmen, mechanics, job-tickets and their working hours are in the system represented by instances of classes with the same names. Their data are processed basically in the same way which is shown on Figure 18. This figure also shows the process of deleting but this function is implemented just for working hours. Only working hours can be deleted without disruption of data consistency in the database because there is no dependency in the database on this entity. Salesmen and mechanics have special workaround which is mentioned later.

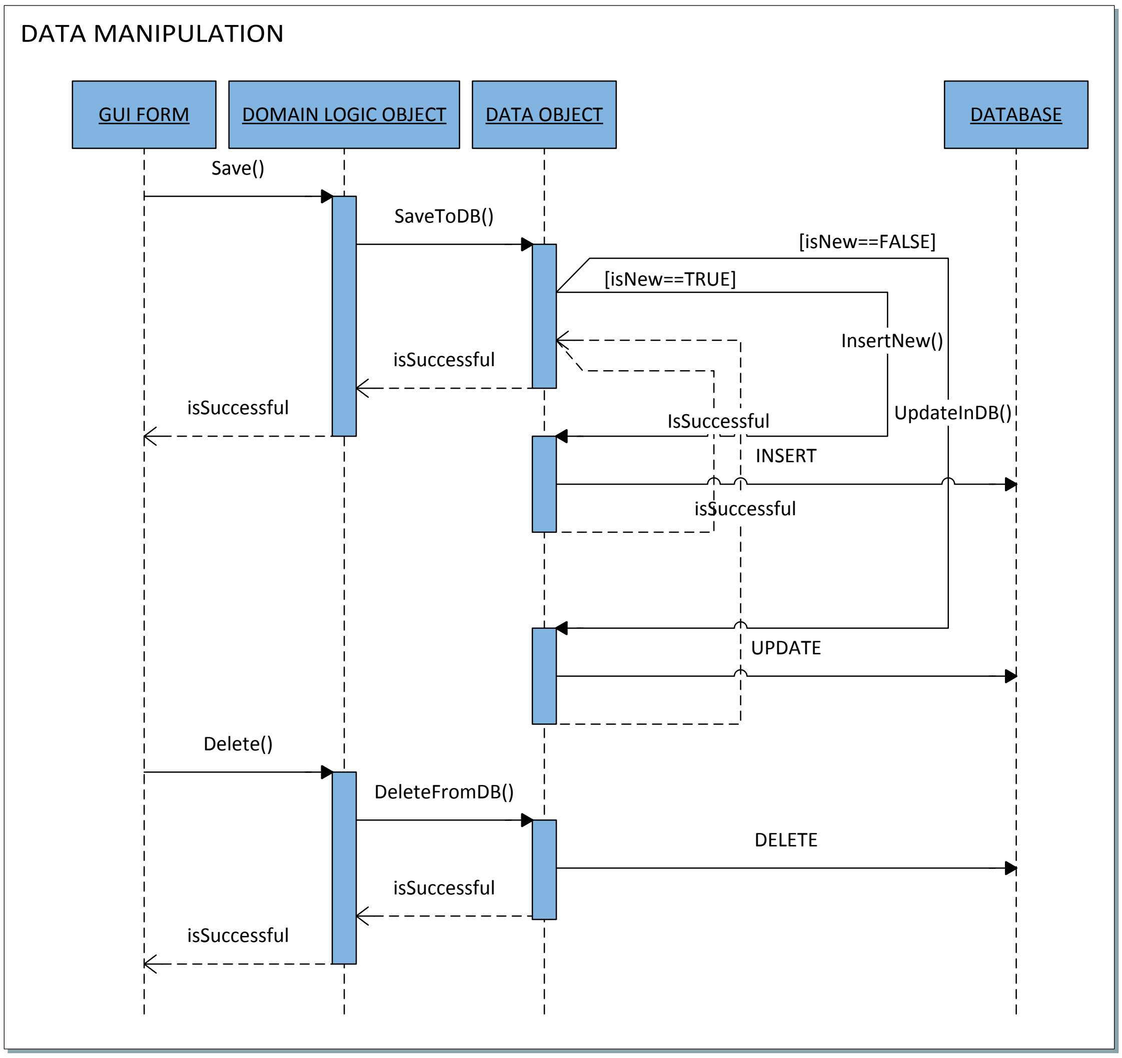


Figure 18 - Data processing

The object which handles all methods and SQL queries for this data manipulation also holds the loaded data. That is why this object is called DO (Data object). The members of the object from the domain logic and Data object from the database layer are shown in Figure 19, in this case for the customer.

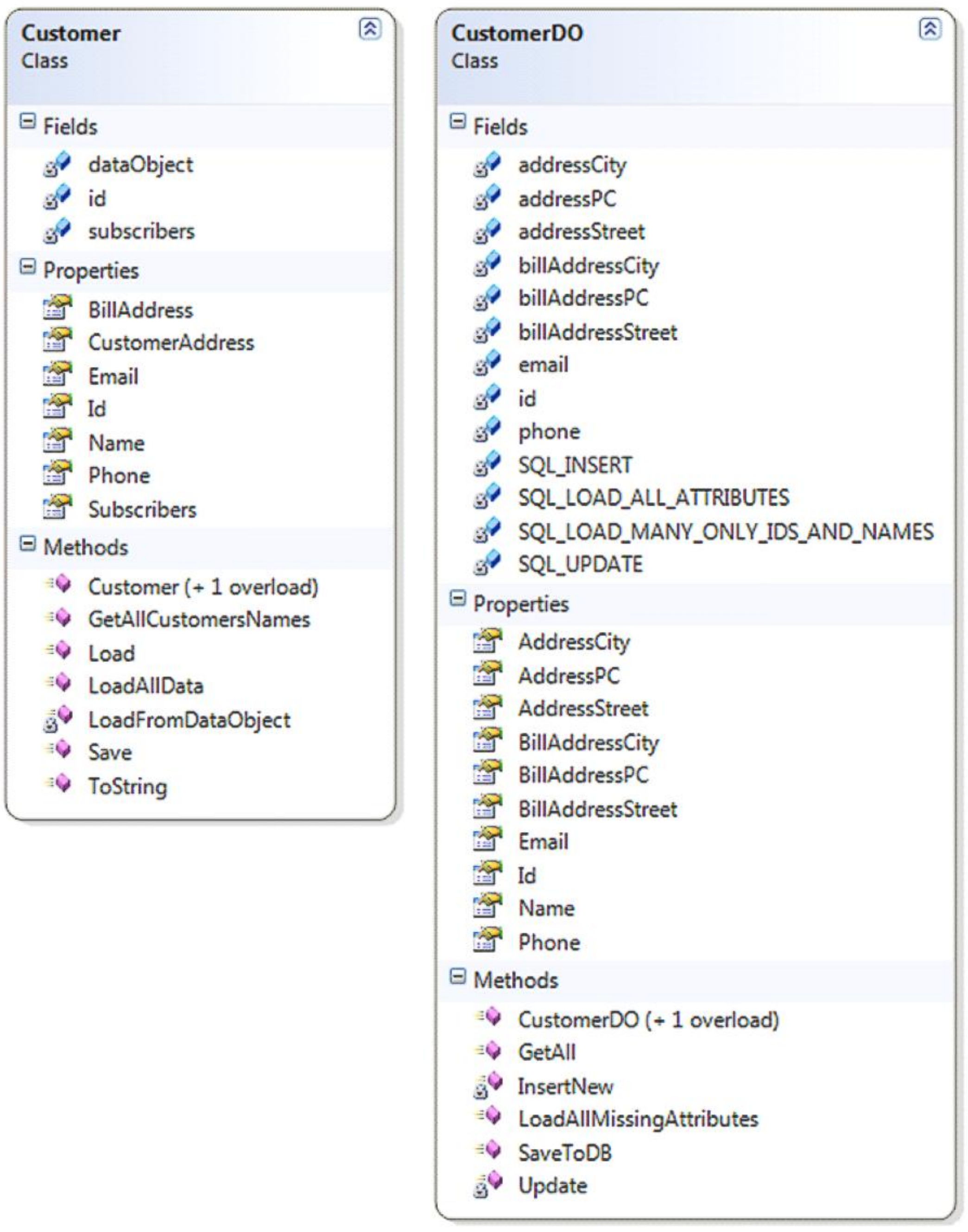


Figure 19 - Data object

This solution was only inspired by Data Mapper design pattern. Mapping itself handles the method of object in the domain logic layer. So in fact this solution is also inspired by Active Record design pattern but with benefits from code separation of Data Mapper. Another advantage of this solution is only one-way dependency in the same direction through all layers of the system.

Every Data object also encapsulates its ID (identifier) loaded from the database. This ID is accessible from outside of the object only for reading and maintains identity between the in-memory object and the database row. Here Identity Field design pattern is used.

5.6.1 Customer

Customers are displayed in two main ways. The first way displays all customers in a list of their names as is shown in Figure 20. For this case ListBox object was used and its property DataSource was set to collection (specifically generic List) of all customers. But this would not be enough if the ToString method would not be override in Customer class. Now the method returns the text string of customer’s name.

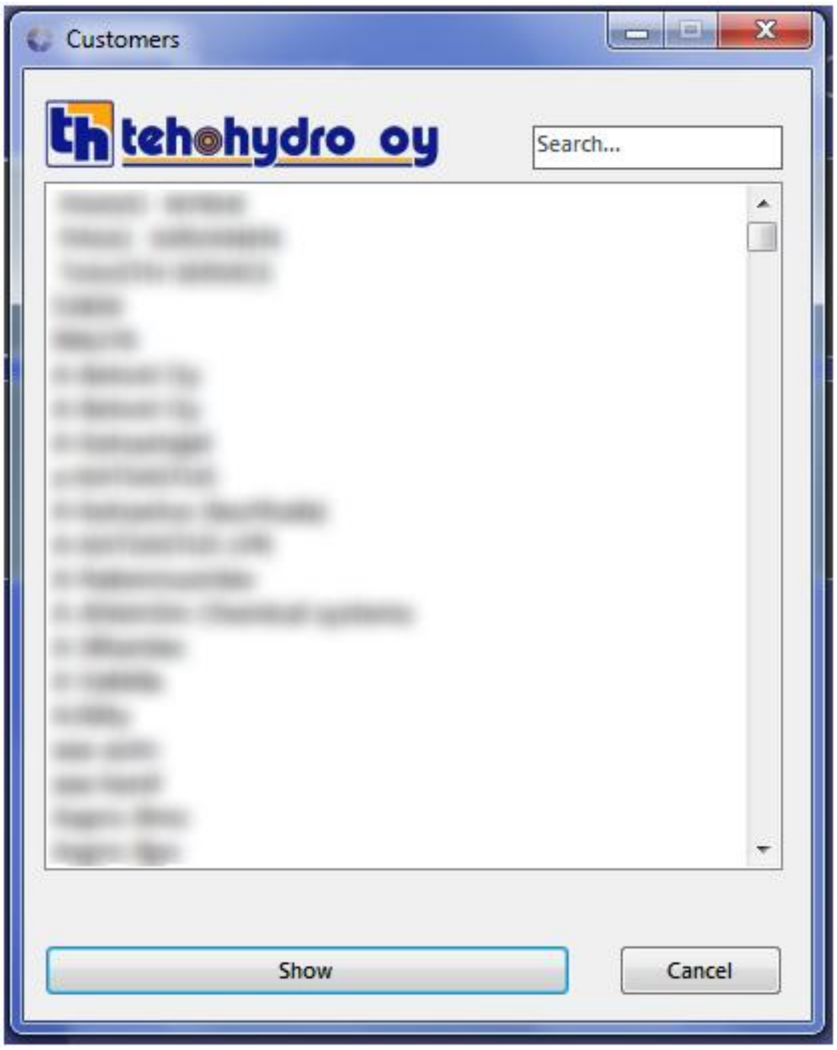
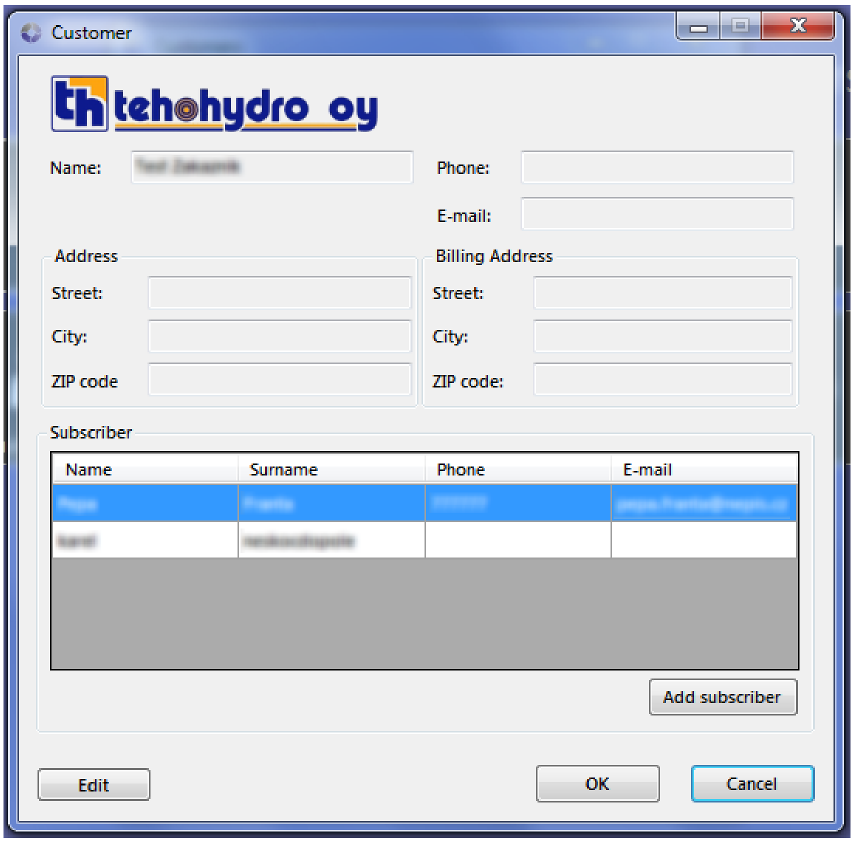


Figure 20 - Customer browse

The second way shows all the details of the customer as well as his subscribers if there are any. This form is shown in Figure 21. For subscribers component DataGridView is used but only its visual part. Filling in the list of subscribers is managed manually. All of this is shown in Figure 21.



Name:	[redacted]
Phone:	
E-mail:	

Address

Street:	
City:	
ZIP code	

Billing Address

Street:	
City:	
ZIP code:	

Subscriber

Name	Surname	Phone	E-mail
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		

Figure 21 - Customer form

To minimize the amount of data transferred from the database all customers are not loaded one by one but all together with only name and ID loaded. All other properties (including collection of subscribers) of one customer are loaded together but only once for each object and only when they are really needed, according to design pattern Lazy load and its type Ghost. (Fowler, 2003)

5.6.2 Salesman

Salesmen are shown in very similar ways as the customer. In case of displaying all salesmen the only difference is in overridden ToString method. For salesmen this method also returns indication whether the salesman is inactive. Inactive indicator is mentioned later. Salesmen are not shown separately because there is no other needed information about salesman than name, surname and ‘inactive’ indication. Figure 22 shows all salesmen. At the first place inactive salesman is shown.

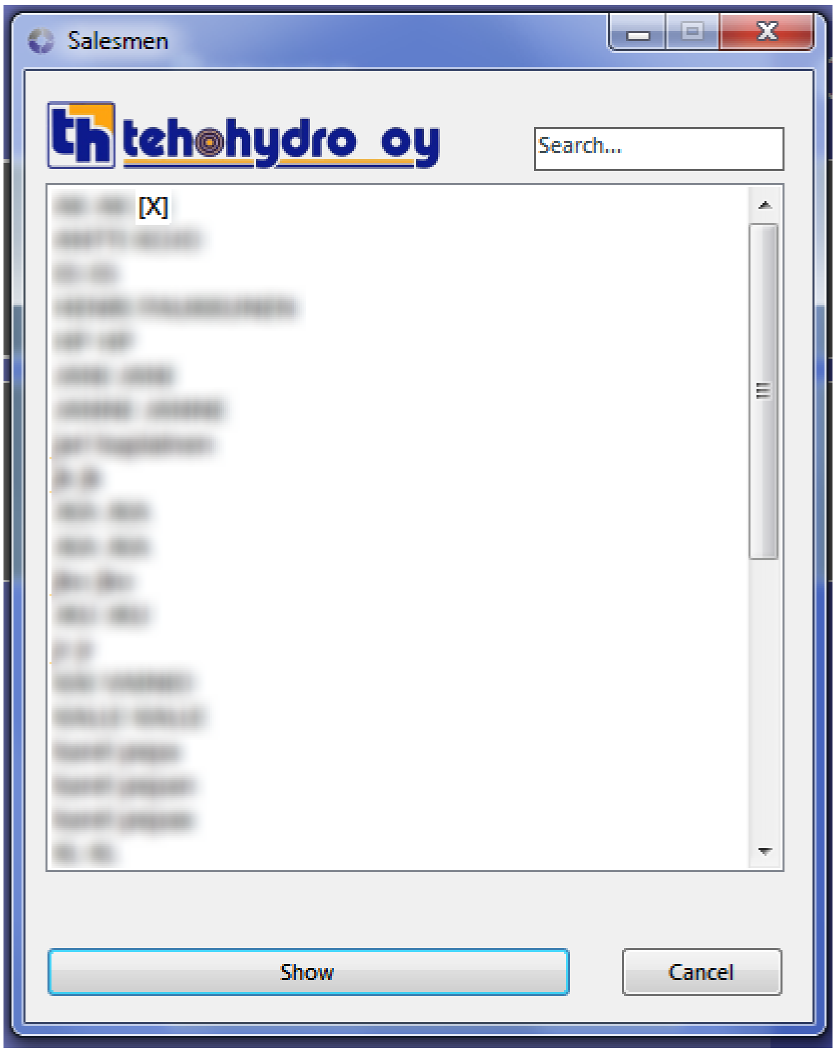


Figure 22 - Form for browsing of all salesmen

Text above mentions inactive state. This is a workaround used instead of deleting the salesmen mentioned earlier. This inactive state causes the salesman`s invisibility in lists of all salesmen during the phase of creating new job-ticket but has no effect on a salesman in already created job-tickets created by him.

For salesman there is no place for Lazy load design pattern because there is all information about salesman needed every time they are shown.

5.6.3 Mechanic

Mechanics are very similar to salesman from the system point of view. Indeed, classes of both objects inherit form the same abstract class Employee as is shown in Figure 23. This inheritance is only on the domain logic layer because Data objects must retain their independence on domain logic.

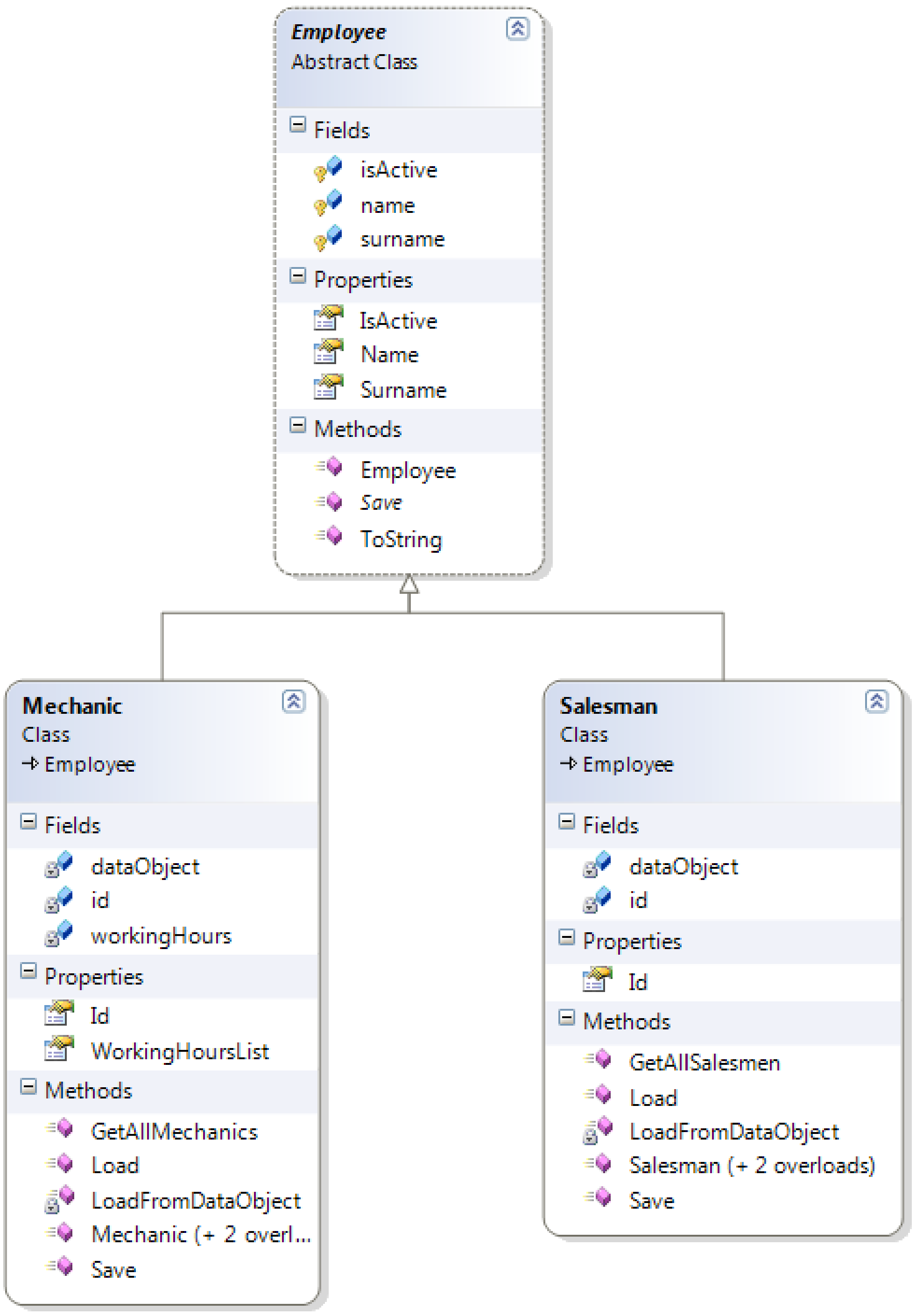
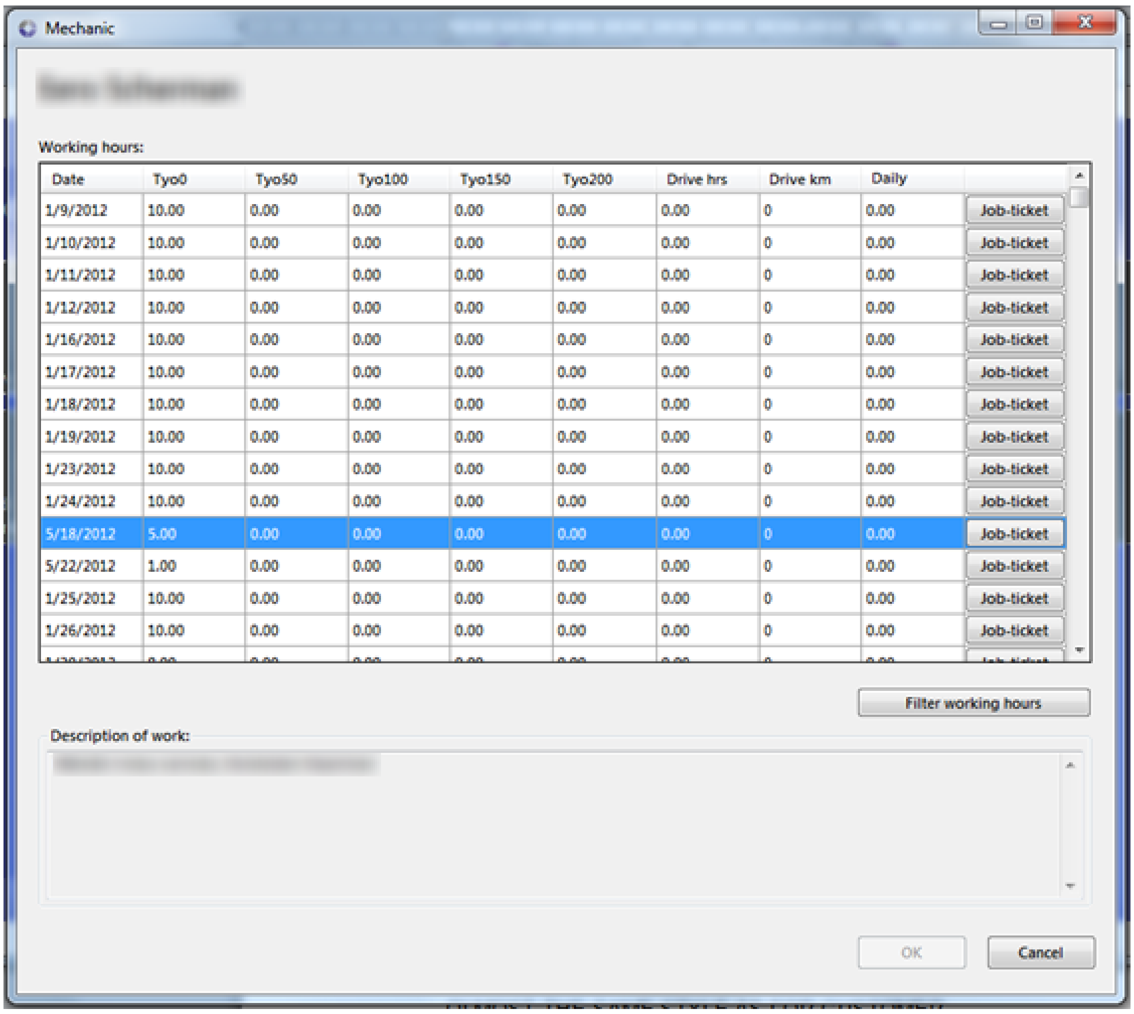


Figure 23 - Employees class diagram

As is evident from Figure 24, mechanic is a little bit more complicated than salesman. That is why the rest of this subchapter is focused on this difference.

From the user point of view the main difference is in the possibility to show details of one mechanic separately. This is due to the need to show also mechanic's working hours, as is shown in Figure 24. Here is evident similarity with customer and his subscribers. In addition to this, there is also need to display description of work for selected working hours.



[redacted]

Working hours:

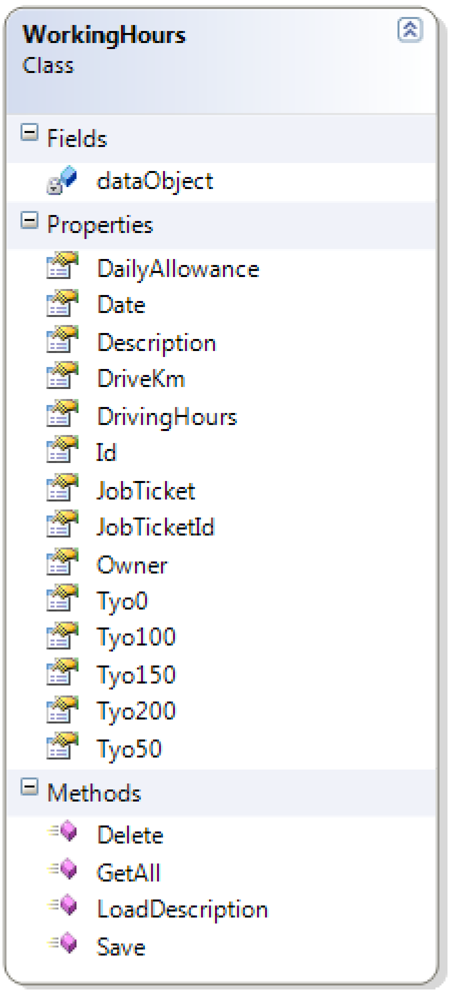
Date	Tyo0	Tyo50	Tyo100	Tyo150	Tyo200	Drive hrs	Drive km	Daily	
1/9/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/10/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/11/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/12/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/16/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/17/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/18/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/19/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/23/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/24/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
5/18/2012	5.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
5/22/2012	1.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/25/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket
1/26/2012	10.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Job-ticket

Description of work:

[redacted]

Figure 24 - Form for mechanic

As mentioned above, a mechanic also encapsulates the collection of his working hours. From implementation point of view working hours are also represented by instances of a special class. All members of this class are shown in Figure 25. Working hours can also be deleted as the only one entity in the whole Job-Ticket system. The reason is already described at the beginning of this chapter.


WorkingHours
Class

Fields
- dataObject

Properties
- DailyAllowance
- Date
- Description
- DriveKm
- DrivingHours
- Id
- JobTicket
- JobTicketId
- Owner
- Tyo0
- Tyo100
- Tyo150
- Tyo200
- Tyo50

Methods
- Delete
- GetAll
- LoadDescription
- Save

Figure 25 - Working hours class

For description of work design pattern Lazy load (its type Ghost) is used. Description of work can be very long. Even infinitely long from database point of view. Strings in .NET cannot have more than 2^{31} characters since property ‘Length’ of the String is a 32-bit integer. (StackOverflow, n.d.) There may also be problems with allocation of memory in heap for that long String. On the other hand so long text string in the whole system is not expected at all. Description is also the only property of working hours which is not needed for showing all working hours.

Whereas the list of working hours should be the largest set in the whole system, filters are also needed. Because filters are one of the most demanding functions there was an effort to use already implemented and tested functions from another part of the system. Filter of job-ticket's working hours is based on filtering of job-tickets themselves. This is the only way how to specify job-tickets for filtering since the ID of the job-ticket is not accessible for the user. Filter for working hours of a specific mechanic is based on both these filters. Figure 26 provides better explanation.

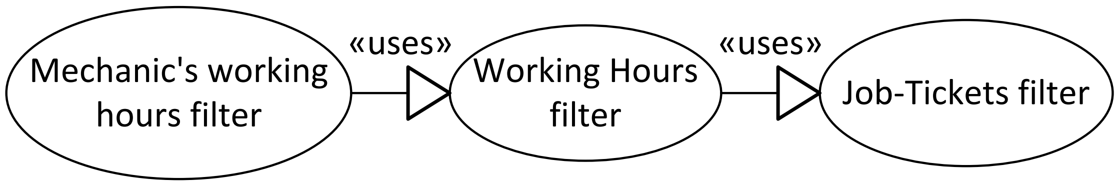


Figure 26 - Filters

All the logic is divided into two parts. The first part is done on the database system level by dynamically created parameterized SQL queries. The second part is solved in an application by filtering generic lists via lambda expressions (MSDN, n.d.) . An example of a used lambda expression can be seen in Figure 27 where working hours (*allFilteredWH*) obtained from filtered job-tickets are filtered and then compared with the collection of working hours of a specific mechanic (*mechanicWH* but also *workingHours*).

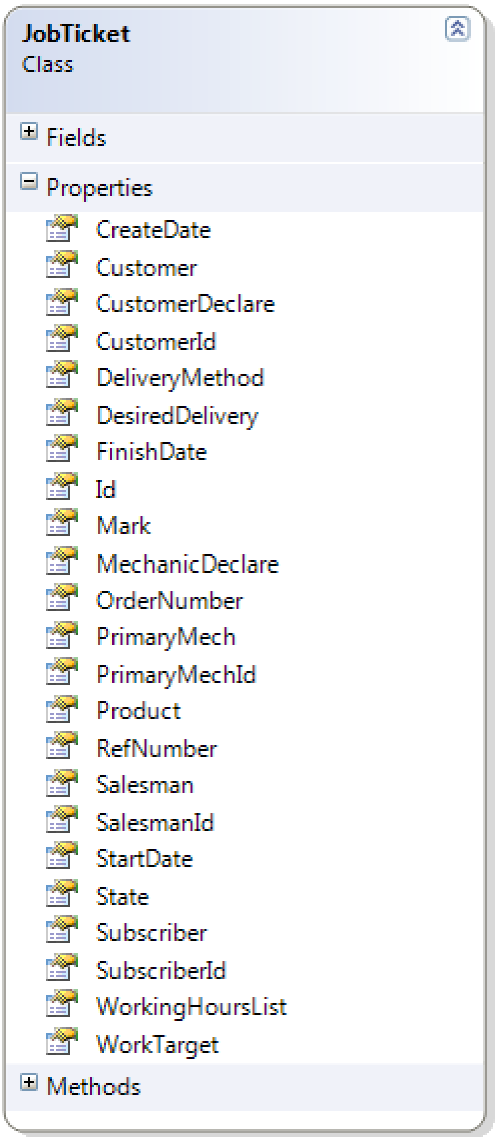
```
this.workingHours = this.workingHours.FindAll
    (
    mechanicWH => allFilteredWH.Exists(loadedWH => mechanicWH.Id == loadedWH.Id)
    );
```

Figure 27 - Lambda expression example

More details about filters are described in the next chapter.

5.6.4 Job-ticket

From Job-Ticket system point of view JobTicket class is the most important part. It is a class where everything is connected together via its IDs (and references) and extended by other information related to job-ticket. More details about job-ticket class are shown in Figure 28.



JobTicket
Class

Fields

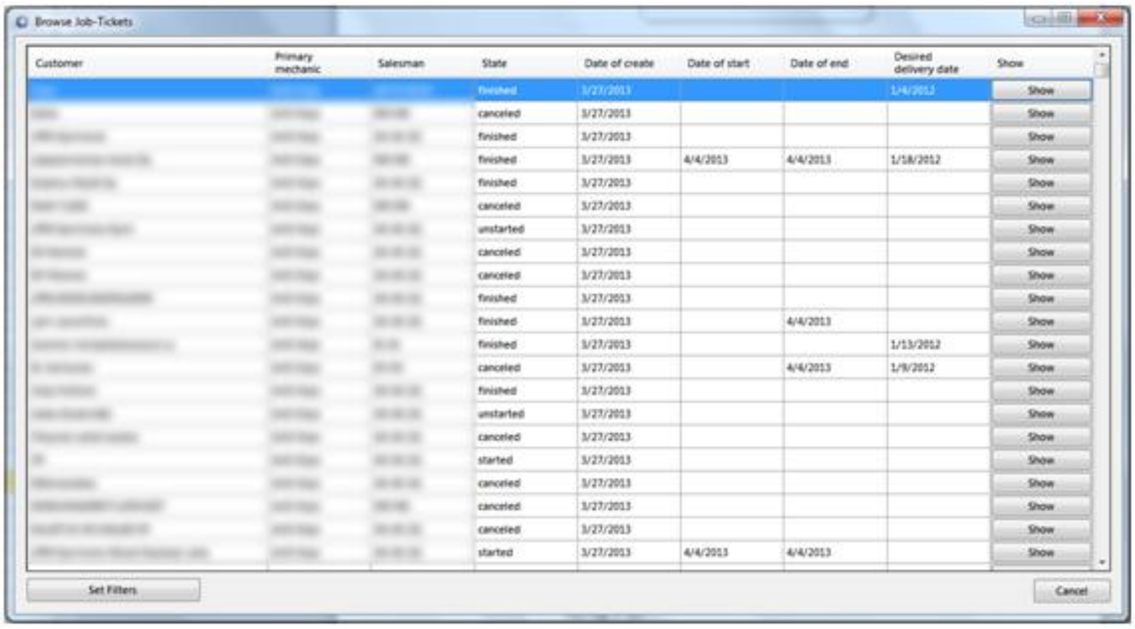
Properties

- CreateDate
- Customer
- CustomerDeclare
- CustomerId
- DeliveryMethod
- DesiredDelivery
- FinishDate
- Id
- Mark
- MechanicDeclare
- OrderNumber
- PrimaryMech
- PrimaryMechId
- Product
- RefNumber
- Salesman
- SalesmanId
- StartDate
- State
- Subscriber
- SubscriberId
- WorkingHoursList
- WorkTarget

Methods

Figure 28 - JobTicket class

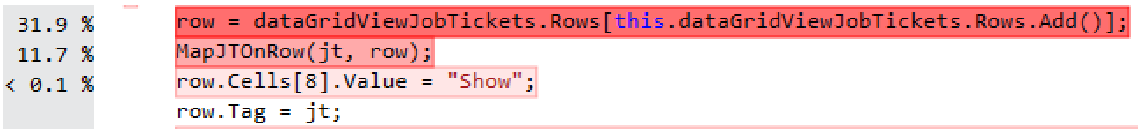
On the presentation layer job-tickets are also shown in two ways as customers or mechanics. The first way displays all job-tickets together in DataGridView component as can be seen in Figure 29.


Browse Job-Tickets

Customer	Primary mechanic	Salesman	State	Date of create	Date of start	Date of end	Desired delivery date	Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013			1/4/2012	Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013				Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013	4/4/2013	4/4/2013	1/18/2012	Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013				Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	unstarted	3/27/2013				Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013				Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013		4/4/2013		Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013			1/13/2012	Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013		4/4/2013	1/9/2012	Show
[redacted]	[redacted]	[redacted]	finished	3/27/2013				Show
[redacted]	[redacted]	[redacted]	unstarted	3/27/2013				Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	started	3/27/2013				Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	canceled	3/27/2013				Show
[redacted]	[redacted]	[redacted]	started	3/27/2013	4/4/2013	4/4/2013		Show

Figure 29 - Browsing of job-tickets

Because of many rows used for better differentiation filling the DataGridView is not so fast. For 5000 job-tickets it takes approximately 5 seconds meanwhile downloading from the database via wireless LAN (Local Area Network) takes about 1,5 seconds. Execution time of query was unmeasurable. The performance analyzator of Visual Studio recognized as the slowest part of code adding a new empty row. The result of this analysis is shown in Figure 30. This adding of rows is implemented in assembly ‘System.Windows.Forms.ni.dll’.



```
 31.9 %     row = dataGridViewJobTickets.Rows[this.dataGridViewJobTickets.Rows.Add()];
 11.7 %     MapJTOnRow(jt, row);
< 0.1 %     row.Cells[8].Value = "Show";
            row.Tag = jt;
```

Figure 30 - Performance analysis in Visual Studio

Due to that problem it was needed to show information about loading for the user. For this purpose ProgessBar and information text about how many items are already loaded were used as shown in Figure 31. Unfortunately it made loading even more slower (approximately 20-30% depending on performance of current computer). Even though everything about loading itself is made in separate thread. This fact is caused by performing steps of ProgressBar and updating text of Label displaying number of processed items. There is no way to improve performance in this case since visual components in .NET are accessible only from the main thread. Setting the visual components from a separate thread is done by Method invoker invoked on the main thread.

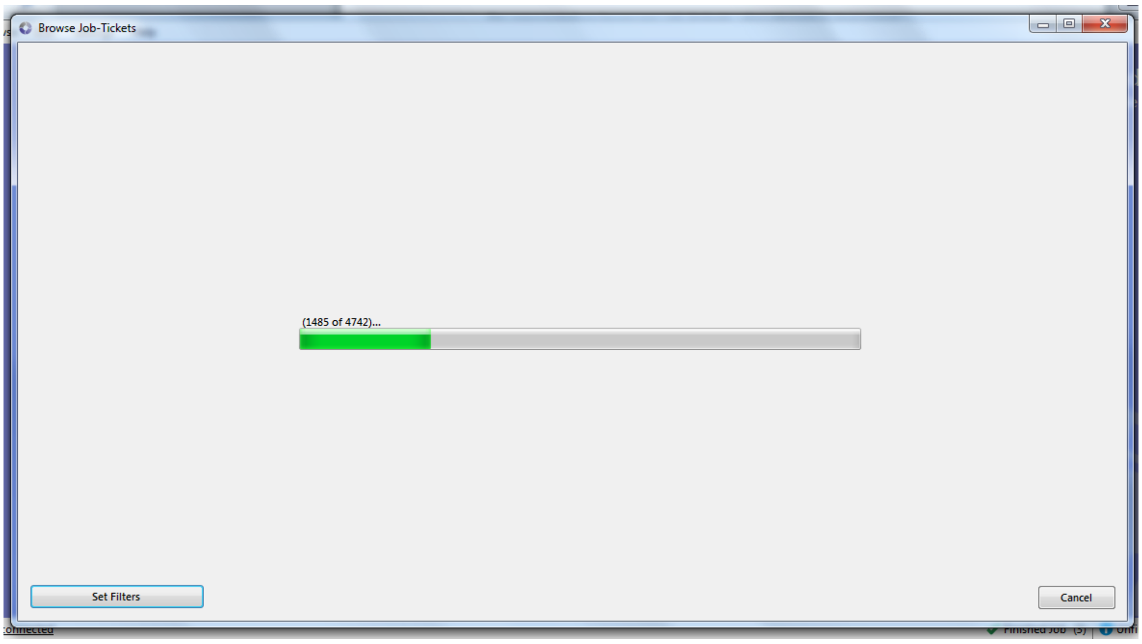


Figure 31 - Loading of job-tickets

The second way for displaying the job-ticket is separately in a special form. The principle is in many ways the same as in the previous cases. All automatically filled fields for the customer are worth mentioning here. Depending on the selected customer in a specific ComboBox also other fields related to the customer of Job-ticket form are filled. This ComboBox itself is prefilled by all customers from the database. The user can select one of the customers by mouse from drop-down list or by typing. During typing first appropriate customer is suggested and other ones are shown under the ComboBox like in Figure 32. Form itself is shown in Figure 33.

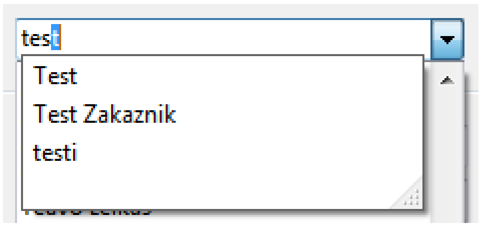
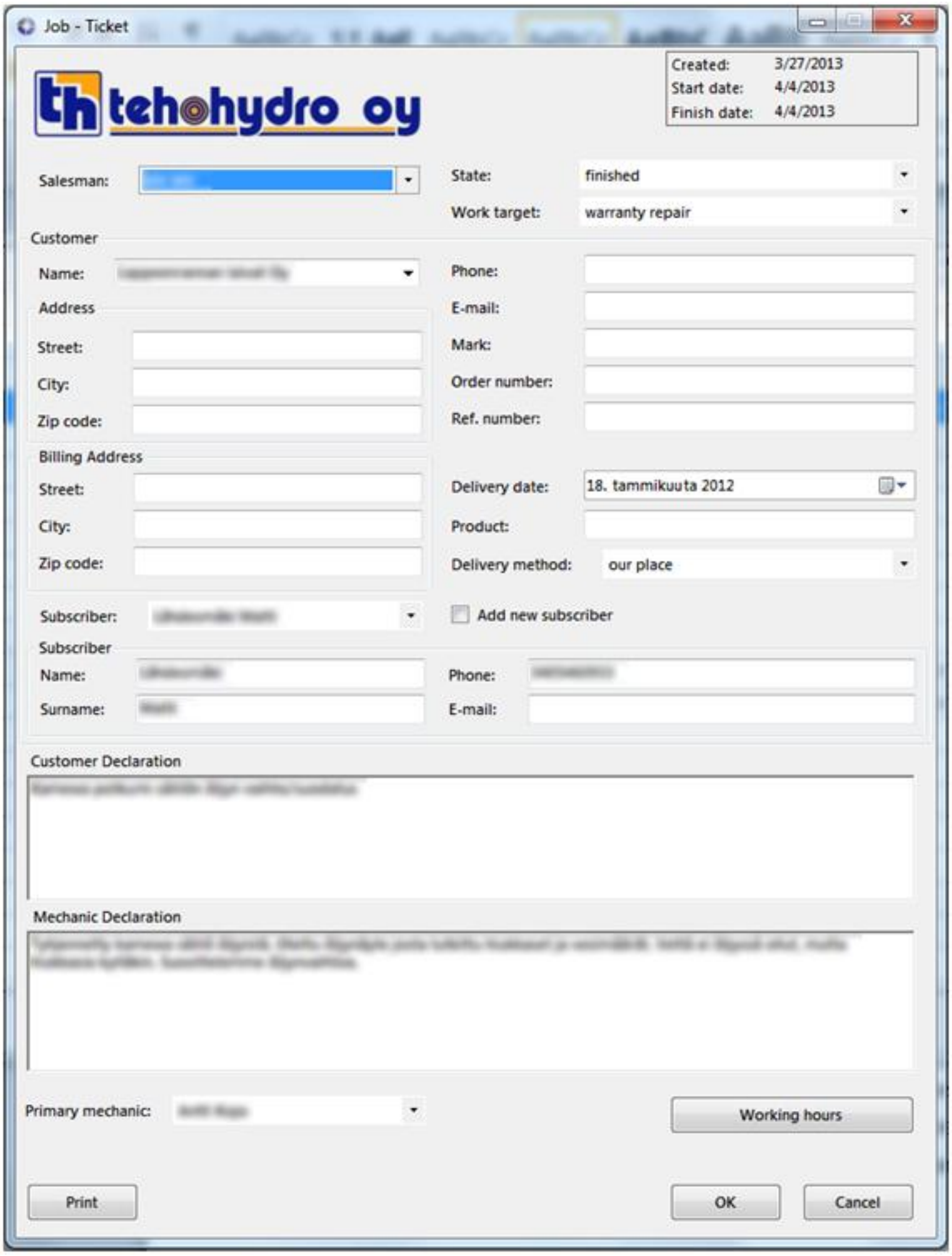


Figure 32 - ComboBox for customer



Job - Ticket

tehohydro oy

Created: 3/27/2013
Start date: 4/4/2013
Finish date: 4/4/2013

Salesman: [redacted]
State: finished
Work target: warranty repair

Customer
Name: [redacted]
Address
Street:
City:
Zip code:
Phone:
E-mail:
Mark:
Order number:
Ref. number:

Billing Address
Street:
City:
Zip code:
Delivery date: 18. tammikuuta 2012
Product:
Delivery method: our place

Subscriber: [redacted]
☐ Add new subscriber

Subscriber
Name: [redacted]
Surname: [redacted]
Phone: [redacted]
E-mail:

Customer Declaration
[redacted]

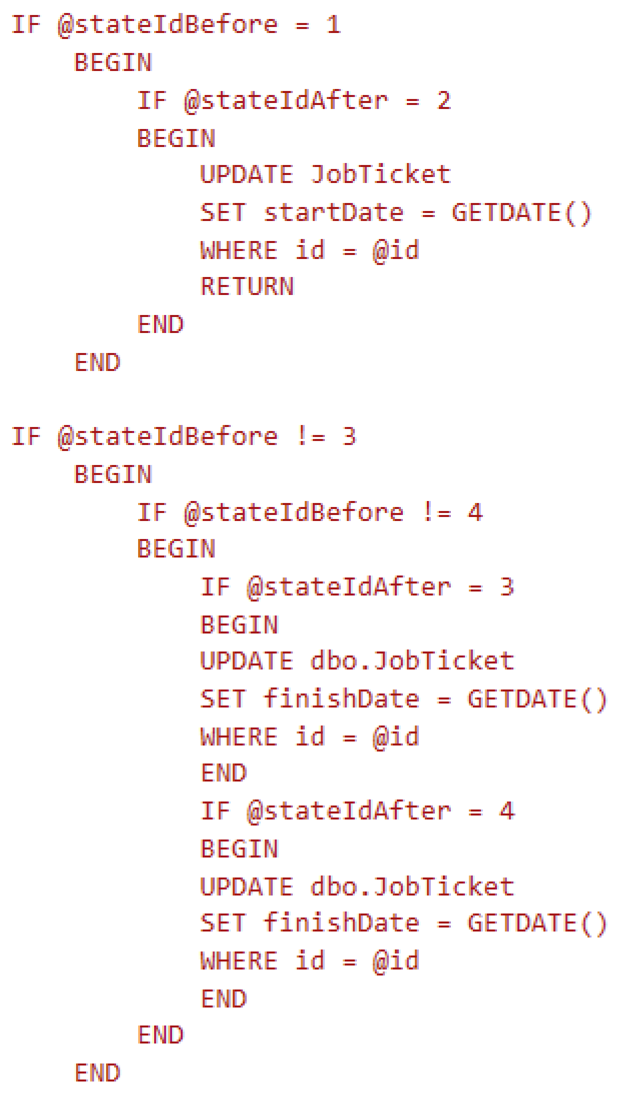
Mechanic Declaration
[redacted]

Primary mechanic: [redacted]

Figure 33 - Job-ticket form

Because creating a new job-ticket also enables to create a new customer with all needed details as well as his subscriber there is a very important way and order how all data are saved. First of all the customer with all his details must be saved. Then his subscriber is saved if there was any. Job-ticket itself is saved as the last one. This is the only right order because for maintenance of references the subscriber is saved with the ID of customer that he belongs to. Job-ticket needs to know the ID of both of them. These IDs are returned by the database system through the call of SCOPE_IDENTITY() method called in SQL command after INSERT statement. This command must be executed as scalar (method ExecuteScalar). If saving of any of those fails, other items in order cannot be saved.

Properties CreateDate, StartDate and FinishDate are set automatically, depending on the previous and current state of job-ticket. This is done by anonymous methods placed in SQL commands. The key part of anonymous method used in an update command is shown in Figure 34. Under the numbers IDs of states from the database table States are hidden. This solution secures that these dates will be set by the same clock, no matter the system time on computers where the instances of Job-Ticket system are installed because those do not have to be synchronized.

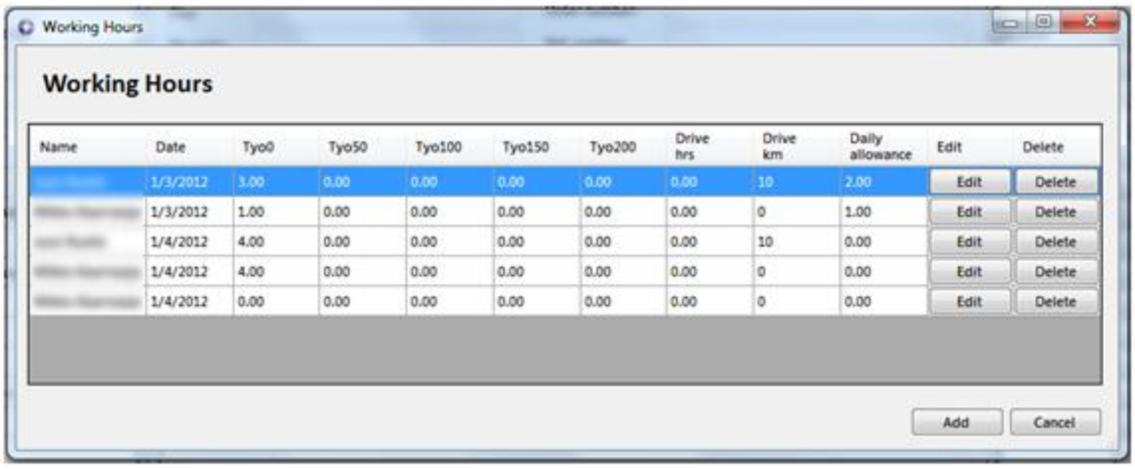


```
IF @stateIdBefore = 1
    BEGIN
        IF @stateIdAfter = 2
        BEGIN
            UPDATE JobTicket
            SET startDate = GETDATE()
            WHERE id = @id
            RETURN
        END
    END

IF @stateIdBefore != 3
    BEGIN
        IF @stateIdBefore != 4
        BEGIN
            IF @stateIdAfter = 3
            BEGIN
            UPDATE dbo.JobTicket
            SET finishDate = GETDATE()
            WHERE id = @id
            END
            IF @stateIdAfter = 4
            BEGIN
            UPDATE dbo.JobTicket
            SET finishDate = GETDATE()
            WHERE id = @id
            END
        END
    END
```

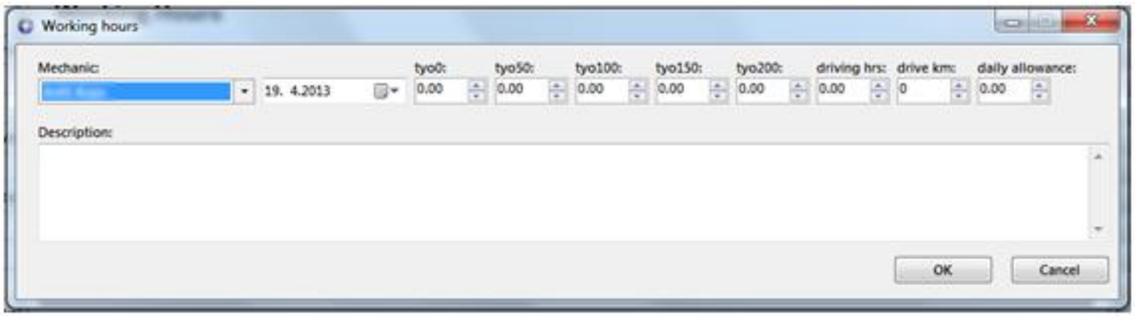
Figure 34 - Anonymous procedure example

Job-ticket is the only place in the system where working hours can be also created, edited and deleted. Showing of the list of working hours is very similar to the way they are shown for a mechanic. The only difference is in buttons making available modifications as shown in Figure 35. The form for editing or creating new working hours is shown in Figure 36 .


Working Hours

Name	Date	Tyo0	Tyo50	Tyo100	Tyo150	Tyo200	Drive hrs	Drive km	Daily allowance	Edit	Delete
[redacted]	1/3/2012	3.00	0.00	0.00	0.00	0.00	0.00	10	2.00	Edit	Delete
[redacted]	1/3/2012	1.00	0.00	0.00	0.00	0.00	0.00	0	1.00	Edit	Delete
[redacted]	1/4/2012	4.00	0.00	0.00	0.00	0.00	0.00	10	0.00	Edit	Delete
[redacted]	1/4/2012	4.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Edit	Delete
[redacted]	1/4/2012	0.00	0.00	0.00	0.00	0.00	0.00	0	0.00	Edit	Delete

Figure 35 - Browse working hours

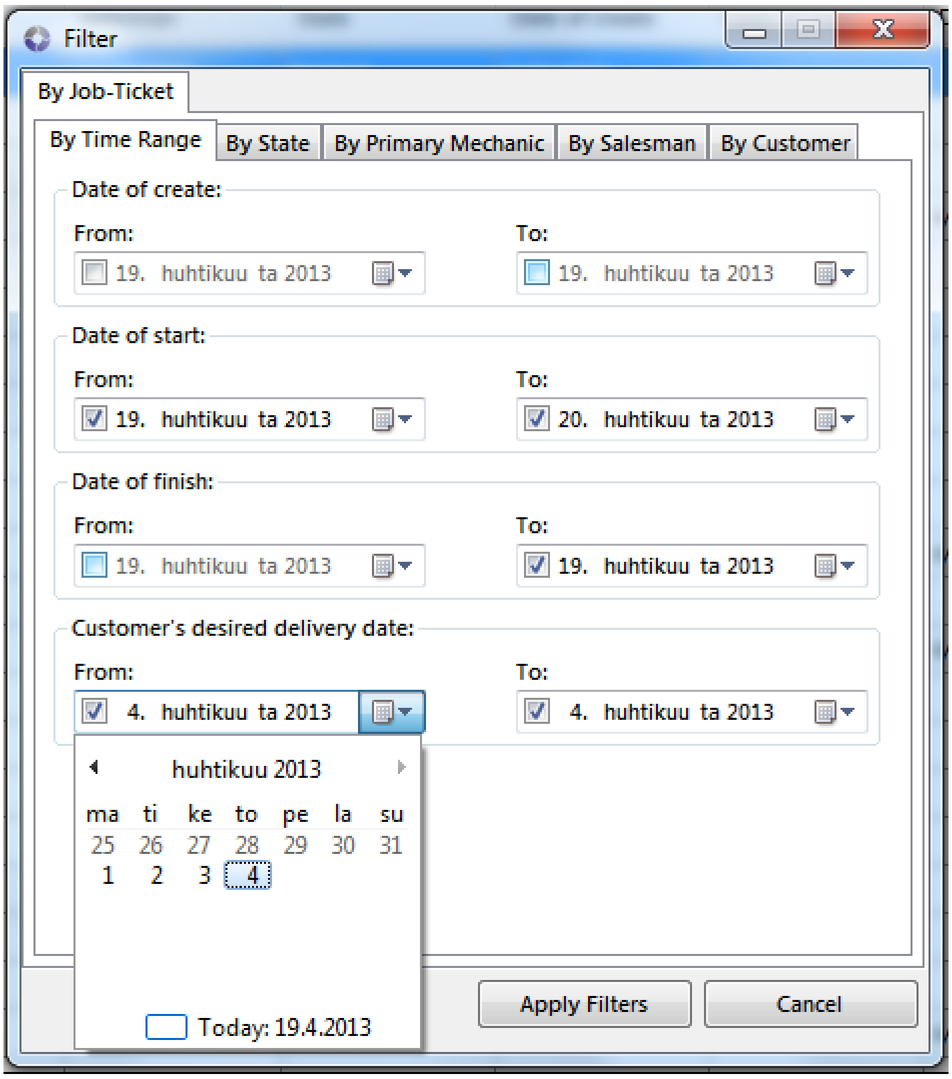

Mechanic:		tyo0:	tyo50:	tyo100:	tyo150:	tyo200:	driving hrs:	drive km:	daily allowance:
[redacted]	19. 4.2013	0.00	0.00	0.00	0.00	0.00	0.00	0	0.00

Description:

Figure 36 - Edit working hours

The filter of job-tickets is undoubtedly the most sophisticated part of the system. Its logic is divided into application and database system. Everything starts by dynamic creating of parameterized SQL query. Here are records filtered by date of creation, date of start, date of finish, desired delivery date and state(s). For all these attributes indexes were set in the database table. On application level there are processed filters by customers, primary mechanics and salesmen. Lambda expressions are used instead of sequential browse of generic collections.

Setting the filter by all kinds of used dates is solved by DateTimePickers with checkboxes. These checkboxes are here because the user can set a range of dates but also one concrete date as is shown in Figure 37. Unchecked whole pair is ignored. In case of a range of dates the system prevents the user to set an empty range. Then the user cannot select an older date ‘from’ then date ‘to’ and vice versa.



Date of create:
From: ☐ 19. huhtikuu ta 2013
To: ☐ 19. huhtikuu ta 2013

Date of start:
From: ☑ 19. huhtikuu ta 2013
To: ☑ 20. huhtikuu ta 2013

Date of finish:
From: ☐ 19. huhtikuu ta 2013
To: ☑ 19. huhtikuu ta 2013

Customer's desired delivery date:
From: ☑ 4. huhtikuu ta 2013
To: ☑ 4. huhtikuu ta 2013

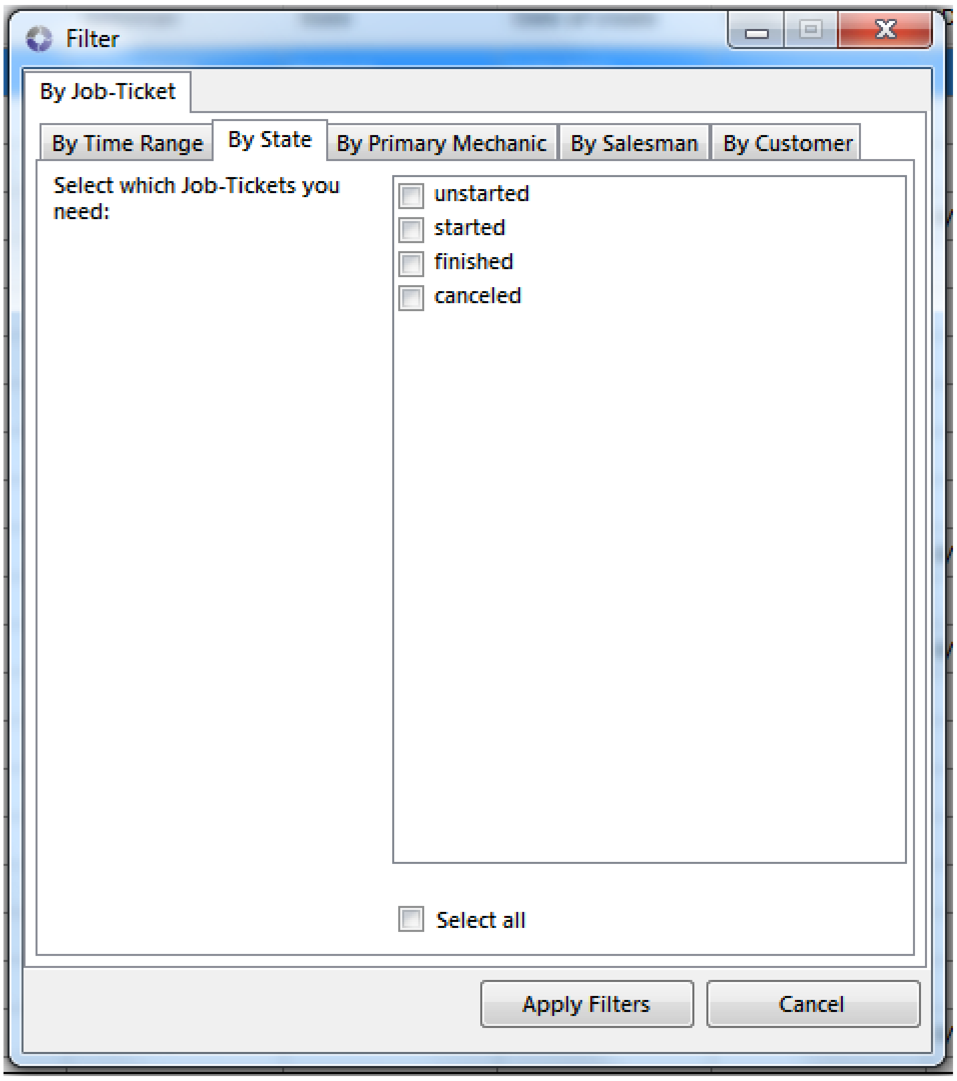
huhtikuu 2013

ma	ti	ke	to	pe	la	su
25	26	27	28	29	30	31
1	2	3	4			

Today: 19.4.2013

Figure 37 - Filtering by time range

Filters by state, primary mechanic, salesman and customer work in the same principle. There is always a list with checkboxes and the user can choose which items should be taken into account. If all or none are selected in a specific category, filter is ignored for this category. An example is shown in Figure 38.



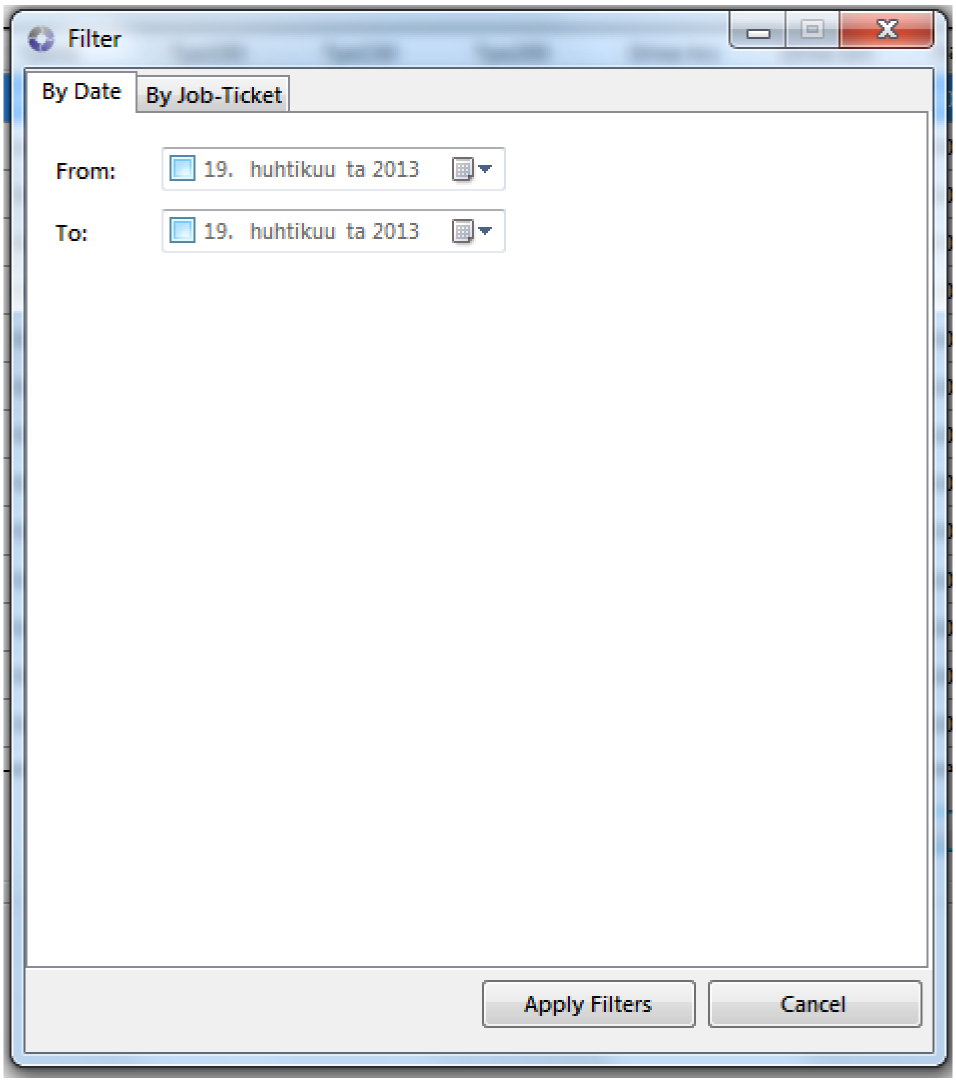
Select which Job-Tickets you need:

- [ ] unstarted
- [ ] started
- [ ] finished
- [ ] canceled

- [ ] Select all

Figure 38 - Filtering by state of job-ticket

Filter for mechanic's working hours is extended by selecting the date range for working hours date shown in Figure 39. A mechanic whose working hours are filtered is taken from a mechanic whose working hours are already shown.



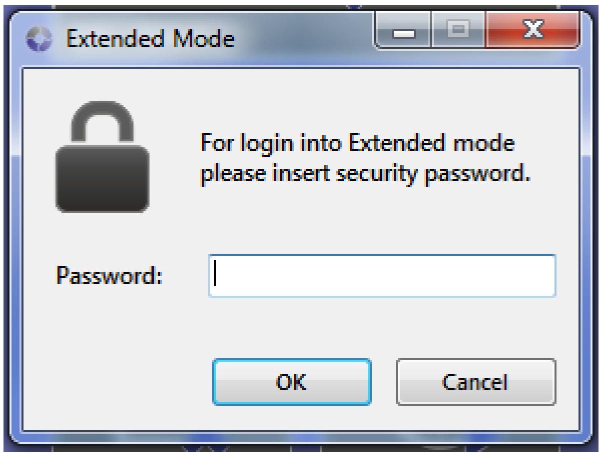
From: 19. huhtikuu ta 2013

To: 19. huhtikuu ta 2013

Figure 39 - Filter of mechanic's working hours

5.6.5 Extended mode

On presentation layer an extended mode is accessible from the right top corner of the main form. Then the user is asked to fill in the password. This can be done in form in Figure 40.

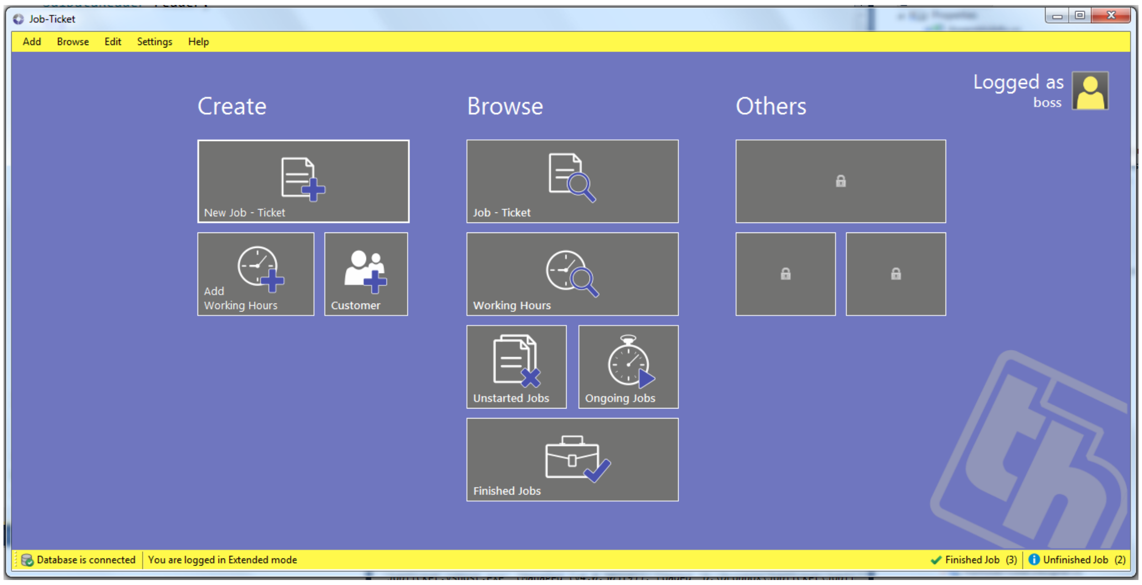

Extended Mode

For login into Extended mode please insert security password.

Password:

Figure 40 - Login form to extended mode

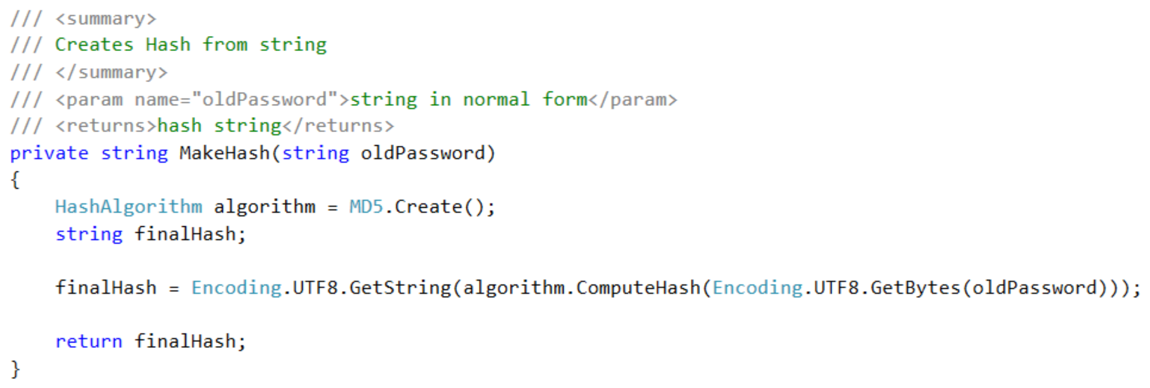
Then the user interface is switched into easily recognizable appearance shown in Figure 41.



Database is connected | You are logged in Extended mode | Finished Job (3) | Unfinished Job (2)

Figure 41 - Extended mode appearance

For password hash it was decided to use MD5 (Message-Digest) algorithm. It is a widely used hash function which creates 128 bit hash value. The method providing this functionality is in Figure 42. This value is saved into a special table in the database.



```
/// <summary>
/// Creates Hash from string
/// </summary>
/// <param name="oldPassword">string in normal form</param>
/// <returns>hash string</returns>
private string MakeHash(string oldPassword)
{
    HashAlgorithm algorithm = MD5.Create();
    string finalHash;

    finalHash = Encoding.UTF8.GetString(algorithm.ComputeHash(Encoding.UTF8.GetBytes(oldPassword)));

    return finalHash;
}
```

Figure 42 - Hash method

5.7 Other Features

This chapter describes all functionalities which do not have any effect to the main functionality of the system but makes the system easier to use, improves its functionality or gives other opportunities how to expand the system with a new functionality.

5.7.1 Language mutations

Requirement for language mutation was known from the very beginning. The system was developed in English language including the user interface. But in the end there had to be a possibility to switch the user interface into Finnish language as well.

For static labels in user interface localization properties provided by Visual Studio in design view of Forms are used. These localization properties are saved in separated resource files for each language mutation and can be edited outside of the whole project or solution. These resource files are usually filled automatically by Visual Studio while editing the form in any of the language mutations but can also be edited or created manually and included into the existing project if the format in these files is followed properly. These resource files are separated for each form. For text strings used in drop-down lists, in message boxes, dynamically changing and so on own resource file common to all these through all forms is used. All these resource files were created only in English language but after that copied and renamed to a correct format for Finnish language. The content of these files was translated by Finnish student

Olli Maaranen. After translation these files were copied back to the folder with other resource files. Switching the language of the user interface is solved by setting the language culture of the whole application, according to the selection of language from the menu by the user.

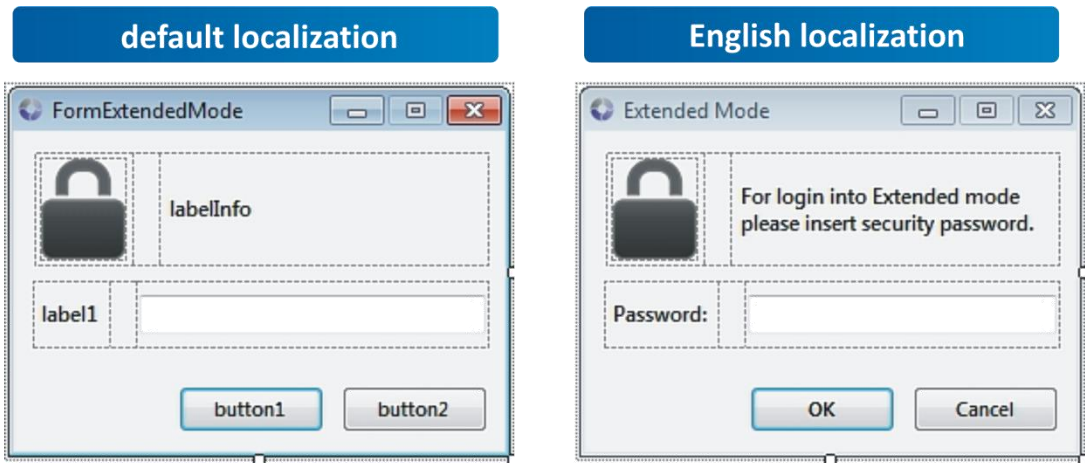
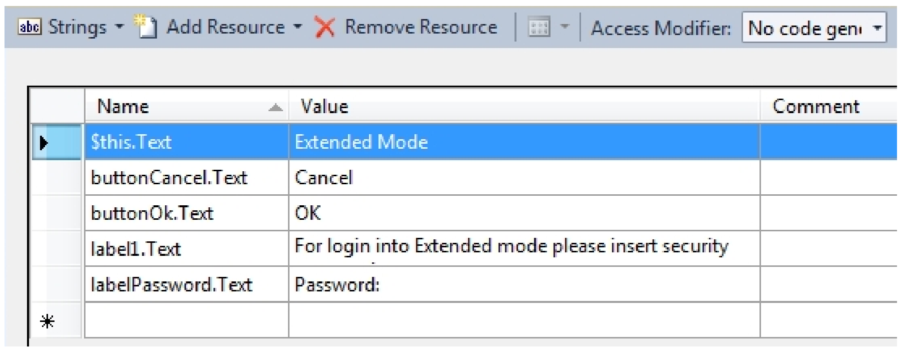


Figure 43 - Localization in MS Visual Studio



Name	Value	Comment
$this.Text	Extended Mode	
buttonCancel.Text	Cancel	
buttonOk.Text	OK	
label1.Text	For login into Extended mode please insert security	
labelPassword.Text	Password:	

Figure 44 - Resource file list

5.7.2 Plug-in support

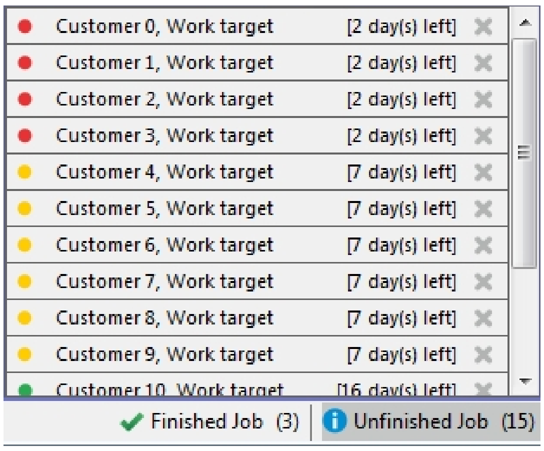
The decision of making the plug-in support was made as response on the impossibility of adding the new functionality to the system without impact of the main application. By impact is meant changing or adding the source code, recompiling the application and re-deployment of the whole system. The main property of the plug-in is integration to the system even though it was created later than the system itself. In our application these improvements are accessible from the main menu in a separate category.

The technical background of this support is strictly designed with focus on easy implementation and isolation of a running plug-in from the main application. Isolation is secured by running plug-in in a separate thread and application domain. Due to that the unstable plug-in will not cause instability of the main application. Implementation of the plug-in is based on a few simple rules, which must be followed for successful integration of the plug-in into the main application of the system. All these rules are included in C# interface, which must be implemented by the main class of the plug-in. This interface was forwarded to the customer during deployment of the system as dll (dynamic-link library) file. Plug-in must be placed in the required folder and those, which do not implement the required interface, are skipped during loading of the plug-in. Execution of the plug-in is made by using the Reflection in .NET technology (MSDN, n.d.)

5.7.3 Notifications

Let us imagine two situations. One or more job-tickets are given. These job-tickets have the set desired delivery date, which will expire in a couple of days. The second situation is about job-ticket when it is not possible to wait for it but somebody is working on it and the point is to need to know as quickly as possible when this job-ticket is done. For both situations notifications are implemented. They should simply aggregate and show job-tickets which are in hurry as well as those which are already done few moments before. These notifications are also something like shortcuts to job-tickets which they are representing. All notifications are shown and accessible from the bottom tool-bar on the main form of the application.

Both kinds of notifications have the same way how they work. This way is based on sending SQL queries to get all the needed data for assessing whether the notification should be raised or not. Notifications are asking for finished or canceled Job-tickets which have been done in the last few hours or minutes. Units and number can be set from settings, as well as refresh rate. Notifications, which are responsible for job-tickets in hurry, are asking for started or not started job-tickets and checking their desired delivery dates. The user can set in-hurry limits in three levels indicating how many days before the desired delivery expiration the notifications should be shown. For the distinction of these three levels in notification colors of traffic lights are used.



🔴	Customer 0, Work target	[2 day(s) left]	✕
🔴	Customer 1, Work target	[2 day(s) left]	✕
🔴	Customer 2, Work target	[2 day(s) left]	✕
🔴	Customer 3, Work target	[2 day(s) left]	✕
🟡	Customer 4, Work target	[7 day(s) left]	✕
🟡	Customer 5, Work target	[7 day(s) left]	✕
🟡	Customer 6, Work target	[7 day(s) left]	✕
🟡	Customer 7, Work target	[7 day(s) left]	✕
🟡	Customer 8, Work target	[7 day(s) left]	✕
🟡	Customer 9, Work target	[7 day(s) left]	✕
🟢	Customer 10, Work target	[16 day(s) left]	✕

✓ Finished Job (3) | Unfinished Job (15)

Figure 45 - Notification area

5.8 Testing and debugging

A project of this size requires advanced methods for debugging. Every function of the system or even every part of the function needs to be debugged and tested to discover and repair as much bugs as possible and minimize the amount of bugs which could get into the next iterations of implementation.

Testing itself was done in three main steps:

1) Testing during or immediately after implementation was performed by person(s) working on functionality which is or was implemented. During that testing only the standard scenario was usually tested and made sure that it works. Every found bug was usually repaired immediately.
2) Testing the development version took place before demonstration and was related to demo application. This testing was focused on stability and optimization and was done usually by all members of the team. Every found error or bug was documented including its simulation process and repaired by creator of the function where the bug was founded.
3) Testing by customer was done at customer's place, in customer's environment and in real situations if possible. This testing required at first deploying of the test version at customer's computer. Bugs were reported

to the team and repaired in the current development version but not in the deployed test application at customer's place.

In the next development version debug reporters will be implemented so testing and reporting of the bugs at customer's place will be much easier for both sides.

There was also need to test the triggers attached to tables in the database. Unfortunately debugging of the triggers would be too time-consuming in comparison with their task so they were replaced by anonymous procedures included into command strings in the application.

5.9 Deployment

Deployment of the test version was usually done by simple ClickOnce applications. This was enough for testing purposes but in the final version there has to be a comfortable and user-friendly way. Not to mention that deployment of the test version required another configuration before the first start. What is more, this configuration required knowledge of the implementation of the system.

The standard installation package will be included in the final version of the application. The user will be able to choose for example where the application should be installed or whether he wants to put a shortcut to his desktop. This application will have its own work folder on known place as well as a configured network-shared folder for data which should be shared between every instance of application installed on computers in the local network. This shared folder will be possible to be set from the application itself.

This kind of deployment expects that there will already be a working Job-ticket database in a network. Fortunately configuration is as simple as was expected. Only snapshot isolation level must be enabled for database instance and that is all. Everything else is set by default. For configuration of the database content a script was created, which takes care of the creating all tables, indexes, relations, data for dictionary tables, etc. This script needs to be run only once but it is also prepared to regenerate the whole database content and start again from scratch with empty tables.

6 Evaluation

Evaluation discusses the results of the work. This chapter describes objectives that were set at the beginning of the work and whether these objectives were successfully met. Evaluation includes also paragraph that explains problems accompanying the development of the system. These problems had not serious character and were resolved. The last subchapter describes tasks that need to be done in the future.

6.1 Meeting the requirements

Dealing with customers during the work progress is a very important part of the entire development. Feedbacks from the meeting with them may help the developers for better understanding what exactly the customer wishes and how the final product should look like overall. That is why numerous meetings with Tehohydro company were arranged to create a system that would be entirely satisfactory to their requirements and wishes. Feedbacks acquired from these meetings made it possible to create a system that was expected.

At the start of the work there was a clear idea about the system, but many essential modifications were made because of these feedbacks. In the end, the customer was impressed by the final solution. All demands were fulfilled and some of them even exceeded the expectations. The customer also appreciated that the environment of the system looks simple and easy for orientation. The idea was to try to keep the placement of user interface components from the previous system for better use orientation so there is no need to read the user manual to start using that. Also Tehohydro's IT specialist liked chosen technical solutions.

6.2 Personal review

The realization of this project accompanied a few small threats, which were sometimes quite difficult to solve, but on the other hand these threats were not of serious nature so the continued development was not compromised. At the beginning the team had three available laptops that were provided by the university. These laptops were very helpful tools in the implementation phase.

One was provided as a special case that offered administrator rights so it was easy to install all necessary software selected by us. SQL server and TFS were running on this special case. The problem was that this laptop had only 2 GB RAM (random-access memory), which was very little, because of too many intensive-processor running programs. There was a problem with frequent paging inside the system. As a result there was a very slow speed of the whole operation system but because this laptop had only a 32-bit operation system, the next extension of the memory was meaningless. (MSDN, n.d.)

The next problem was connected with merging of the source code by using TFS. The problem can occur in the situation, when a programmer keeps his source code long time, does not upload his code version often and makes important changes in the code. Then the problems with merging the code may appear. The solution of these problems was found in using history library, where all previous source code versions are recorded. This library is included in TFS for purposes of losing some important code. .

6.3 What needs to be done

Because the work will still continue after submission of this thesis, there are a few programmed issues that need to be done. The whole system needs to have localization of the user interface in Finnish language. This part of work is important because the current system is translated into Finnish language so localization in the new system is necessary for this language as well. Another task is adding of the notification function that shows job-tickets which are in hurry as well as those which are already done. This functionality is useful, because it should serve like a shortcut for each job-ticket. And finally there is need to create an installation package, which creates a local folder where all important files for running the system will be created (and also stored). All these tasks will be done next month.

7 Summary and discussion

Information systems are one of the important parts of information technology. These sets of technological resources and methods serve data processing and data storage. The purpose is to present information for user's needs. This thesis described the design and solution for the new information system dealing with administration and management inside Tehohydro company. The system was named Job-Ticket and kept all important functionalities from the previous old system but it was programmed by using completely different technologies. Moreover this system is extended with new features, which were the wishes of the employees in order to simplify the work. Finally, a few features were added by us to improve the system for more convenient use and for the possibility of continued expansion of this system in the future.

The development was conducted with Microsoft technologies. We were developing the solution in a work team and we were sharing the actual version of the project through the Team Foundation Server provided by Microsoft Visual Studio. During the development of the system we had numerous meetings with the customer (Tehohydro representatives) and also with our supervisor. On these meetings we presented what we had done so far and discussed the next opportunities that would be possible to implement. This kind of feedback helped us also to find out errors and remove them. We were lucky in dealing with Tehohydro company, because their IT specialist, Mr. Jari Harju, was able to provide us all necessary information and he helped us with the deployment of our system into praxis.

The system was subjected to testing by using corporate data provided by the company. The first functional version was deployed into praxis in one meeting. This version was fully functional without some less important details, which are also included in the implementation plan. This is the current state, where we actually are. We are planning to have the last meeting with Tehohydro at the beginning of May, where we will deploy our final version of the application.

This work gave us many experiences. We have learned how to organize work in a team. Each team member knew what his role is and what his job description is as well. For this purpose there was a very powerful tool called OneNote. In the implementation phase we used our previous experiences from home university but we also worked with new technologies that were not familiar to us before. These experiences we consider to be the greatest contribution to us. Due to the fact that we had many meetings with Tehohydro company, we learned how to clearly present our ideas to the customer, how to deal with him and how to gain his feedback into our side as well. We have realized that exactly this part is one of the most important of the whole process because a satisfied customer represents the image of the product quality.

References

Barashev, D. & Thomas, A., n.d. *Gantt Project.* [Online]
Available at: http://www.ganttproject.biz/learn
[Accessed 18 4 2013].

Blue Claw Database Design, n.d. *Comparison of Microsoft Access vs. Other Database Systems.* [Online]
Available at: http://www.blueclaw-db.com/access_versus/
[Accessed 18 4 2013].

Codd, E. F., 1969. *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks,* s.l.: IBM Research Report.

European Commision, 2005. *Impact Assesssment of Regional & National E-Business Policies.* [Online]
Available at:
http://ec.europa.eu/enterprise/newsroom/cf/_getdocument.cfm?doc_id=5350
[Accessed 18 4 2013].

Fowler, M., 2003. *Patterns of Enterprise Application Architecture.* s.l.:Addison-Wesley.

Lasater, C. G., 2010. *Design Patterns.* s.l.:Jones & Bartlett Publishers.

Microsoft Corporation, n.d. *Microsoft Access.* [Online]
Available at: http://office.microsoft.com/en-us/access/
[Accessed 18 4 2013].

Microsoft Corporation, n.d. *MS OneNote.* [Online]
Available at: http://office.microsoft.com/en-us/onenote/
[Accessed 18 4 2013].

Microsoft Corporation, n.d. *MSDN.* [Online]
Available at: http://msdn.microsoft.com/en-us/cc136611
[Accessed 18 4 2013].

Microsoft Corporation, n.d. *Team Foundation Server.* [Online]
Available at: http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx
[Accessed 18 4 2013].

MSDN, n.d. *Lambda Expressions - Programming Guide.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/bb397687.aspx
[Accessed 18 4 2013].

MSDN, n.d. *Memory Limits for Windows Releases.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/aa366778.aspx
[Accessed 18 4 2013].

MSDN, n.d. *Reflection - Programming Guide.* [Online]
Available at: http://msdn.microsoft.com/en-

us/library/ms173183%28v=vs.80%29.aspx
[Accessed 18 4 2013].

Paisley University, 2003. *Is there a need for information systems?.* [Online]
Available at: http://www.aylos.com/Default.aspx?tabid=76
[Accessed 18 4 2013].

Prezi Inc., n.d. *Prezi Manual/FAQ.* [Online]
Available at: https://prezi.zendesk.com/forums
[Accessed 18 4 2013].

StackOverflow, n.d. *StackOverflow.com.* [Online]
Available at: http://stackoverflow.com/questions/140468/what-is-the-maximum-possible-length-of-a-net-string

Tehohydro Oy, n.d. *Tehohydro Oy.* [Online]
Available at:
http://www.hydrauliikka.eu/index.php?sivu=yritys&alasivu=henkilosto
[Accessed 18 4 2013].

Figures