Managed Operations Software (MOS)

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Department of Electrical and Information Technology

Bachelor thesis:
Rukhaya Alkuraiti
Kristina Kadar

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2012

Abstract

This thesis is a report from a project where the web based, platform independent application ‘MOS’ - Managed Operating Software was developed. The question asked was: ‘How can we help DISA's customers to get a simple overview over the maintenance of their machines and to plan said maintenance?’
MOS was created specifically for DISA, which is an international company providing innovative casting technology.
The application was written using Java and JSP whereas the database was created using SQLite. Two of the functions included are scheduling maintenance and a calendar. The user accesses MOS via a simple web page. This report documents the creation of MOS and its strong sides and weaknesses. It is the hope of the authors that this thesis will bring insight to those wishing to learn more about MOS, JSP or any other components involved.

Keywords: MOS, DISA, JSP, Java, SQLite

Sammanfattning

Denna avhandling är en rapport från ett projekt där det webbaserade, plattformsoberoende programmet ”MOS” - Managed Operations Software utvecklades. Den fråga som ställdes var: ”Hur kan vi hjälpa DISAs kunder att få en enkel överblick över underhållet av sina maskiner och planera detta underhåll?”
MOS skapades speciellt för DISA som är ett internationellt företag som tillhandahåller innovativ gjutningsteknik.
Plattformen utvecklades i Java och JSP medan databasen skapades med SQLite. Några av de implementerade funktionerna är schemaläggning för underhåll, tillgång till kalender och så vidare. Användaren kommer enkelt åt MOS via en webbsida.
Denna rapport dokumenterar skapandet av MOS och dess starka sidor och svagheter. Författarna hoppas på att denna uppsats kommer att kunna ge insikt till dem som vill lära sig mer om MOS, JSP eller andra berörda ämnen.

Nyckelord: MOS, DISA, JSP, Java, SQLite

Foreword

As students in such a wide field as computer engineering it is almost impossible to predict what you will end up doing at the end of your studies. When the opportunity presented itself to carry out a Bachelors project at DISA we were immediately interested as we had come to know DISA to be a large company with a leading position in its field. As the project proposal was very interesting we decided to apply.
Throughout this project we have done our best to produce a satisfactory solution to what has turned out to be a demanding but intriguing problem.

We would like to thank DISA for this opportunity, as well as our supervisor Nils Assarsson.

Helsingborg, June 2012
Ruhkaya Alkuraiti & Kristina Kadar

List of contents

1 Introduction 1
1.1 Background 1
1.1.1 DISA 1
1.1.2 DISA’s products 1
1.2 Purpose 2
1.3 Starting point 3
1.4 Requirements specification 3
1.5 Limitations 5
1.6 System description 5

2 Tools 7
2.1 Programming language 7
2.1.1 JSP 7
2.1.1.1 Syntax 8
2.1.1.2 Pros 9
2.1.1.3 Cons 9
2.1.1.4 Usage 9
2.1.2 PHP 9
2.1.2.1 Syntax 10
2.1.2.2 Pros 11
2.1.2.3 Cons 11
2.1.2.4 Usage 11
2.2 Database 11
2.2.1 SQLite 12
2.3 Server 13
2.4 Dropbox 14
2.5 Eclipse EE 14
2.6 SQLite Manager 14

3 The working process 15
3.1 Information gathering 15
3.2 Implementation 15
3.3 Testing 16
3.4 Documentation 16

4 MOS 17
4.1 The Java part 19
4.1.1 Db.java 19
4.1.2 SqliteDb.java 20
4.1.3 CheckItem.java 22
4.1.4 OrderItem.java 22
4.1.5 User.java 22

4.2 The JSP part 23
4.2.1 Login 24
4.2.2 Home 25
4.2.3 Order 25
4.2.4 Employees 26
4.2.5 Maintenance 27
4.2.6 Help 27
4.3 The database 29
4.4 Other elements 31
4.4.1 Style.css 31
4.4.2 Internationalization 31
4.4.3 Calendar 32
5 Results 34
5.1 Screenshots 37
5.1.1 The login screen 37
5.1.2 The home screen 38
5.1.3 The employee screen 39
5.1.4 The order screen 40
5.1.5 The maintenance screen 41
5.1.6 The help screen 42
6 Conclusions and comments 43
6.1 Future development 44
7 Dictionary 45
8 References 46
8.1 Image source 47

1 Introduction

1.1 Background

1.1.1 DISA

DISA is one of the leading manufacturers and suppliers of innovative metal casting technology [1]. They currently produce moulding and shot blast systems which are sold to customers worldwide. These machines are used to manufacture everything from small car parts to large furnaces.

The company itself dates back to the early twentieth century. It was first founded in the year 1900 and for the first 37 years DISA manufactured light machine guns. Throughout the years the company has produced a variety of products including sewing machines, tractors and petrol pumps. Since the early 1960's however, DISA is entirely committed to the manufacturing of metal casting technology.

DISA is based in the town of Herlev outside Copenhagen, Denmark. There are also over 30 sites worldwide including manufacturing and sales agents. DISA currently holds 97 active patents and has to this day installed a total of 2200 moulding machines.

1.1.2 DISA’s products

The moulding equipment available to DISA's customers is either vertical, called DISAMATIC©, or horizontal, called DISA FLEX©. The DISAMATIC was first introduced in 1964. Apart from the moulding equipment DISA also provides several accessories such as various transport systems which can be connected to the equipment in question. Service and support of different kinds are also available.

The different solutions offer specialized adaptation to the customer's needs. For example, DISAMATIC offers a fast vertical production of up to 510 moulds per hour while DISA MATCH offers an efficient production of smaller product series where the moulding pattern needs to be altered frequently. These are only two of the many moulding solutions offered by DISA.

Figure 1. DISAMATIC

1.2 Purpose

From many years of experience DISA has noticed that their customers often have issues with keeping track of the different maintenance tasks to be performed on their equipment. Following the maintenance routines is important because not only does it extend the lifespan of the equipment but it also prevents them from having to be taken offline for unscheduled repairs when they break. These emergency repairs usually cost more for the company than a scheduled repair because loss of production and the risk that the broken part has caused damage to additional parts. In light of this, DISA would like to be able to provide a platform on which these issues are addressed and solved.

The question we asked was: How can we help DISA's customers to get a simple overview over the maintenance of their machines and to plan said maintenance?

The purpose of this project was to implement a web based, platform independent tool for this purpose. This thesis describes this platform and the work behind it.

1.3 Starting point

This project started originally in the autumn of 2011 as a part of a project course. The group that worked on it then did as much as they could in the short duration of the course. What they accomplished then was the basis of MOS. Since then a lot of things have been reworked or added by us.

What is left of the earlier version is the user interface, the files concerning employees, the help tab, Db.java, Users.java, SqliteDb.java and mms.sqlite. The latter is the database file. The latter two have been modified extensively. A more detailed description of the changes made follow in this report.

In addition to the program from the previous project, we also had a list of requirements that we put together following our first meeting with DISA. Our goal was to meet all the demands, but because of the projects time restrictions, we had to ignore certain points. However, we saw that the requirements removed would not affect the program's functionality.

Below is the list of the requirements compiled. The chapter ‘Results’ contains a description of the requirements that were not met and the reason for not meeting them.

1.4 Requirements specification

R1 Check-list

1. Ability to add entries
2. Ability to remove entries
3. Ability to edit entries
4. Ability to change time scope.
5. Ability to scroll through the check-list's pages using next/previous buttons.

6. Contents of the checklist are inserted / updated in the calendar.

R2 Planning calendar

1. The calendar should show one month at a time.
2. Clicking on a day should reveal a more detailed picture of the tasks to be performed on that date.
3. The calendar should show the data in different colours depending on the task's priority.
4. Adding other tasks apart from those present in the check-list should be possible
5. Ability to reschedule a task.
6. Ability to remove a task.
7. Ability to add a time at which the task is to be performed.

R3 Resources

1. Ability to see available employees.
2. Ability to see available spare parts.

R4 Priority levels

1. There are three different levels: urgent, semi-urgent, not urgent.
2. The calendar represents every level with a colour; urgent - red, semi-urgent - orange, not urgent – yellow.

R5 Language support

1. The program should have support for multiple languages.
2. Upon delivery the program should support the following languages: English and Swedish.
3. The language support should be easy to extend with additional languages without having to re-engineer the entire platform.

R6 Security

1. The program should be protected against SQL injections.

R7 Reporting

1. It should be possible to bring up a particular service report.

2. It should not be possible to edit a service report.

1.5 Limitations

Because of the time constraints not all desired functions could be implemented. One of these was unfortunately security. In the version of MOS delivered to DISA there is no security implemented except for a user name and password that are not encrypted. Functionality of the platform was prioritized instead.

1.6 System description

Upon delivery, MOS will include the features that we concluded, together with DISA, that their clients are in need of in order to keep track of the maintenance of their equipment.

MOS was developed for DISA's behalf. However, DISA will not have any direct use of any of the parts included in MOS. It is their customers around the world who will hopefully find good use of it. DISA will be the owner of the software and the retail rights.

Viewed from DISA's customers' perspective MOS is a program with which they can keep track of when maintenance of equipment should be carried out in order to be able to maximize use of this equipment.

With the aid of MOS, DISA's customers will be able to keep track of the maintenance work to be done during the day as this is the first thing you see after logging into the program. You can by clicking on a maintenance job get a more detailed description of the work to be performed, who will carry it out and more. Moreover, one can see what maintenance work needs to be performed during the coming period or a little later in time as MOS allows customers to search through all the registered orders. As previously mentioned, one can again obtain a detailed description of each maintenance work.

In addition to keeping track of maintenance work, DISA's customers using MOS will be able to keep track of all employees as there will be a tab in the program designed for this. With the aid of this tab you may obtain a detailed picture of the company's employees and their information such as name, skills and more.

2 Tools

2.1 Programming language

Since the software developed in this project had to be platform independent the choice stood primarily between PHP and JSP.

Even though PHP has benefits like being beginner friendly and being supported by almost all webhosting platforms JSP was chosen for this project. The weak types and lax syntax check of PHP makes it less secure than JSP. Since the software was going to initially be run locally the webhosting factor did not really matter at this time. Also, PHP is very well suited for less complex projects while JSP is better suited for more complex projects due to the standard classes and its scalability.

2.1.1 JSP

JSP is an abbreviation for JavaServer Pages. It is used to create dynamic web pages where, for example, HTML or XML makes up the static content. The dynamic content is made up of JSP elements which are marked with special tags in the code. JSP uses the Java programming language and is owned by Sun Microsystems.

JSP makes it easier to develop web based software as it separates the logistics part from the design and visualisation as well as supports reusable, component based design. The designer and programmer can work individually without having to rely on each other. In other words; JSP separates the Java code that creates content from the HTML code that presents it to the users [2]. It is also much more accessible compared to normal servlets which for example invites page designers to participate in the development [3].

A JSP page becomes a standard Java servlet when compiled. A servlet is a platform independent server-side module which can be used to enhance the functionality of a web server without demands of extensive maintenance or support. This servlet is cached and re-used until the original JSP page is modified.

2.1.1.1 Syntax

The JSP syntax can be divided into directives, declarations, expressions and scriptlets.

Directives tell the JSP engine how to handle the page, but generally does not generate output. They are always embedded in the `<%@ ... %>` tag. Several directives exist and the most frequently used are page and include. Any number of page directives can exist in a single JSP page. For example, the code `<%@ page import="java.util.*" buffer="16k"%>` loads all the types declared in java.util and sets the page buffer to 16k.

The include directive makes it possible to separate content into smaller elements, making them easier to handle. The code `<%@ include file="test.html" %>` includes the content of test.html into the JSP page.

Declarations are used to declare or save methods or information within the JSP page. For example, the code `<%! int i=0; %>` would be a valid declaration. Note that a '!' has replaced the '@' found in a directive.

Expressions are typically used to evaluate a Java expression, convert it to a string and include it to the output JSP page. For example, the variable integer 'I' declared in the previous item could be displayed with `<%= i %>`.

Scriptlets, or JSP code fragments are embedded within `<% ... %>` tags. Here it is possible to put virtually any Java code, for example this snippet:

```
<% for (int i=1; i<=4; i++) {%>
<% Hello world! %>
<% } %>
```

This will give the following output:

```
Hello world!
Hello world!
Hello world!
Hello world!
```

As a side note, comments in JSP pages are put within `<%-- --%>` tags. As opposed to regular HTML comments they are invisible to users even if they view the page source.

2.1.1.2 Pros

- JSP has very strict syntax check and it is therefore harder to make undetected mistakes [4].
- It is platform and server independent.
- Easily scalable.
- Good portability. JSP goes by the rule: Write once, run anywhere.

2.1.1.3 Cons

- The learning curve is considered to be quite steep, especially if the programmer is inexperienced. Knowing Java may help somewhat.
- Since JSP is not as widely used as for example PHP both the community and the documentation is scarcer.
- Not all webhosts can host JSP based web pages. Therefore the hosting costs for JSP is considered higher than for PHP.

2.1.1.4 Usage

JSP is primarily used in projects that require higher security. Because of the fact that JSP uses several layers of authentication it creates an environment with a higher security. It is also used in complex projects that contain multiple layers. One of the companies using JSP for their services is Altavista.

2.1.2 PHP

PHP is an abbreviation for Hypertext Preprocessor. It is a widely used general-purpose server side scripting language released under the PHP License as free software used to create dynamic web content. The PHP code is interpreted by a PHP processor module on a web server, creating the web page [5]. PHP can be deployed on most web servers or as a standalone shell on most operating systems.

The inventor of PHP in 1995 was Rasmus Lerdorf, a Danish/Canadian programmer. The PHP group [5] is now in charge of the language, as well as serve as the formal reference. PHP is generally a safe language to use if the best practice programming rules are applied. Most security issues arise when users deviate from this practice [6].

There are security add-ons for PHP, for example PHPIDS, which protects against intrusions such as SQL Injections, DoS and cross-site scripting [7].

2.1.2.1 Syntax

PHP code can be embedded directly into HTML code. As PHP is a high-level language its syntax is similar to languages such as C & C++. The processing of the code is made easy by simply encapsulating it in tags within the HTML code itself. There are several tags available, the most common being `<?php ... ?>`. Within these tags, any PHP code may be entered [5].

For example, the following code: `<?php echo 'Hello World'; ?>` will output the classical ‘Hello World’ statement.
It is also possible to write functions, such as:

```
<?php function fooFunction() { return 'awesome';}
echo 'PHP is ' . fooFunction() . '!'; ?>
```

will output ‘PHP is awesome!’.

PHP 5 supports complete object-orientations as well as private member variables and methods, abstract classes and methods as well as final methods. An example is a PHP class could be:

```
class Person {
public $firstName;
public $lastName;
public function __construct($firstName, $lastName = '') {
$this->firstName = $firstName;
$this->lastName = $lastName;
}
```

2.1.2.2 Pros

- PHP is considered easy to learn even for non-developers due to its dynamic typing [8].
- A large community and extensive documentation is available.
- Almost all webhosts can host PHP applications.
- PHP is Open Source.

2.1.2.3 Cons

- Since it is a dynamically typed language it is easy to make mistakes and the code may be hard to debug.
- Best practices have to be followed to ensure security and easier maintenance.
- PHP works best for simpler projects as the code easily gets bloated as the complexity increases.

2.1.2.4 Usage

PHP is widely used for web development since it is easy to learn and use. As is it server based, installation procedures require only a minimum of technical knowledge. Alas, as the typing is so forgiving it is easy to get carried away and ending up shooting oneself in the foot. If best practices and code standards are not followed the result can easily end up being ‘hacks on hacks and a whole lot of crappy spaghetti code’[8].

PHP is used for web projects of all sizes and by literately everyone, from novices to experienced professionals. Even people who do not know any other programming language take up PHP. Some notable projects that make use of PHP are Joomla, Drupal, Wikipedia, Facebook and WordPress.

2.2 Database

In addition to selecting which programming language we would use we also needed to determine which database manager we would use throughout the project. The choice was between SQLite and MySQL, mainly because they are the most widely used and well documented relational databases.

After studying each manager we compared them with each other and studied which one would be appropriate. We came to the conclusion that SQLite was the appropriate handler for this project.

The arguments for the choice of SQLite instead of MySQL is that SQLite is much easier to install than MySQL [9]. This is good as in many cases the user of MOS may have no experience in this field. In addition, MySQL goes under the GNU General Public License and therefore a paid license is needed to use MySQL which is not the case with SQLite as it is in the public domain.

As we found out that users probably will not use databases that exceed 1 GB SQLite will suffice as that is the limit of its storage capacity. Additionally, DISA's customers will initially have their database installed locally. As SQLite is stored as a single file and, as mentioned earlier, does not need complicated installation procedures it is preferable to MySQL in this case. The single file also makes it easy for the user if the database was to be moved to another computer.

2.2.1 SQLite

SQLite is a small and lightweight relational database management system [10]. It is one of the most widespread database solutions for client storage in web browsers and operating systems, as well as cellphones and MP3 players. A few notable examples of companies utilizing SQLite in their products are Apple, Google, Airbus and McAfee.

SQLite was designed by D. Ropert Hipp in the year 2000 when working to design software to be used on U.S. warships where there was a need to allow software to access a database without having a database management system installed. Its source code is now public domain and it is continuously upgraded and tested. Some of the many useful features resulting from the extensive testing are that data transactions are fully ACID-compliant and adheres to most standard SQL syntax. ACID implies that all database transactions are Atomic, Consistent, Isolated, and Durable [11]. This means that all changes to the database are ‘all or nothing’. Either the whole change is applied or the database remains unaffected. This acts as a safe guard towards unexpected errors such as power failures and program crashes.

SQLite is the library-of-choice in many applications due to its size. A full-featured SQLite database is usually less than 350 KiB and a more sparsely furnished database may even be smaller than 100 KiB. This database is stored as a single, platform independent file on the host computer. The database handles several read operations simultaneously whereas only one write operation is allowed at any given time.

2.3 Server

Apache Tomcat was chosen as the web server [12]. It is an open source and platform independent web server, not to be confused with the 'Apache Web Server'. Apache is a C implementation of a HTTP web server whereas Apache Tomcat runs on Java and supports Java Server Pages, JSP. It is light-weight and easy to set up.

Apache Tomcat, or Tomcat for short, comes with three components; Catalina, Coyote and Jasper. Catalina is the so called servlet container which implements the Sun Microsystem specification for servlets and JSP [13]. Coyote is the HTTP connector built into Tomcat. It manages incoming connections via TCP ports and sends requests to the Tomcat engine. Jasper is the JSP engine which compiles JSP files to Java code. These files can then be processed by Catalina.

The choice to use Tomcat as a server was mainly because we had previous experience in working with it and it would serve our purpose well. This meant that we could save a lot of time when it came to problem solving and trouble shooting.

As the choice of Tomcat was in a sense obvious we did not investigate other available options.

2.4 Dropbox

To easily share files while working Dropbox 1.4.7 was used [14]. Dropbox is a service provided by Dropbox, Inc. and offers cloud storage and file synchronisation. A program can be installed on the computer and then it automatically synchronises the files in a selected folder with the Dropbox server.

2.5 Eclipse EE

Since the project was a dynamic web project it was developed in Eclipse EE 3.7 (Indigo) [15]. In the rest of the thesis it will only be referred to as ‘Eclipse’.

2.6 SQLite Manager

SQLite Manager 0.7.7 was used to create and manage the SQLite database used by the platform [16]. SQLite Manager is an extension for Mozilla Firefox. It does not install its own SQLite library but uses the one installed in Firefox. In this case it was version 3.7.10.

3 The working process

In order to obtain a clear picture of what to deliver to DISA at the end of the project a meeting was scheduled at DISA's main office in Denmark. This meeting resulted in a list, clearly showing DISA's requirements, i.e. what the MOS needed to be able to do.

As we were two people working on this project we divided the work into two parts. Each of us then had the responsibility of satisfying the demands in one of the parts. Despite of this we used one and the same work method where we collected information, implemented and tested in parallel throughout the project. This lies close to the ‘Do Whatever’ model, as we felt it necessary to use a very agile model in a project such as this. Moreover, it is similar to the working method that was used during the project course in the fall of 2011.

To make it possible to work on the project simultaneously from different locations we created a Dropbox account. This is mentioned in section 2.4.

3.1 Information gathering

Neither one of us had any prior experience working with JSP which meant that the main part of the starting phase was devoted to learning this language. However, we were constantly forced to gather additional information during the course of the project when we encountered various problems.

Information was sought on various forums for developers who have experience in working with JSP and websites for the various programs that were used during the project. These are mentioned in Chapter 2.

3.2 Implementation

After gathering enough information about the usage of JSP the implementation of MOS commenced. MOS consists of several small parts that are described in Chapter 4, ‘MOS’. These parts are mainly divided in the different functions of MOS. Therefore the implementation took the form of small iterative phases which included collection of information, implementation, and finally testing.

3.3 Testing

After each completed stage everything implemented during that phase was tested. In cases where the part implemented had a connection to the parts previously implemented and thus could affect these parts, testing of all the parts connected was carried out.

A final test was carried out at the end of the project. All elements included in MOS were tested to ensure that the product delivered to DISA was working.

3.4 Documentation

The plan was that the documentation would be done in parallel with the project. Unfortunately this was not met as the project for the greater part is a development-type project which makes it difficult to document anything at all until the program is near completion. However, there were some parts such as when we decided if we were to develop in PHP or JSP which were recorded at the beginning. Thus there was documentation done in the beginning and end of the project.

4 MOS

MOS is composed of three parts: a part that covers the Java files, the second consisting of the JSP files, and a third and final part that is the database.

All parts are assembled in one package in Eclipse, named disa. This package is what will ultimately be delivered to DISA with its report.

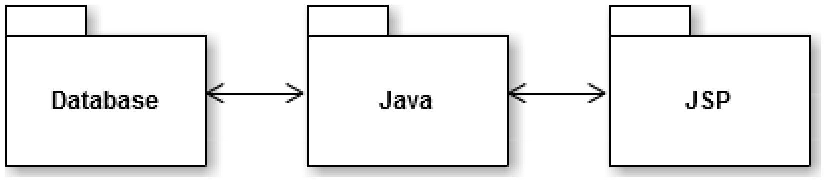


Figure 2. The structure of the platform showing the three parts: the database, the Java part and the JSP part. The flow of information is indicated by the arrows. As can be seen the JSP part does not communicate with the database directly.

The reason for designing MOS in this manner comes from DISA's need for web-based software as well as our own desire to create a platform independent program. Since DISA's customers are based across the world and have various technical skills these seemed like sound choices.

Having a web-based program helps the customers as they do not need to install any software on their own computers. They will also have continuous access to the very latest version simply by using their browser to access DISA's web server.

Making the software platform independent makes updates and further development simpler as it does not matter which operating system the developers are using.

During the development of this software the web server Apache Tomcat Server was used for testing from the development environment, Eclipse.

To fulfil DISA's demand for a web-based program, we used HTML and CSS to develop the layout of the web page (the static content). For the dynamic content, JSP was used. JSP uses regular Java commands to send or retrieve information from a web page. This lead us to create Java files for the different classes used in the program.

A more detailed description on each Java file can be found in section 4.1.

As the program sends and retrieves data a database is needed to store it. We used the simple and lightweight database SQLite which is stored on the user's computer in the form of a single file with file ending .sqlite.

In the same way as the Java files and JSP files exchange data with each other when you access the website the Java files and the database file exchange information to retrieve or send the data needed by the JSP files.

4.1 The Java part

This section covers a certain number of classes needed for a functioning program.

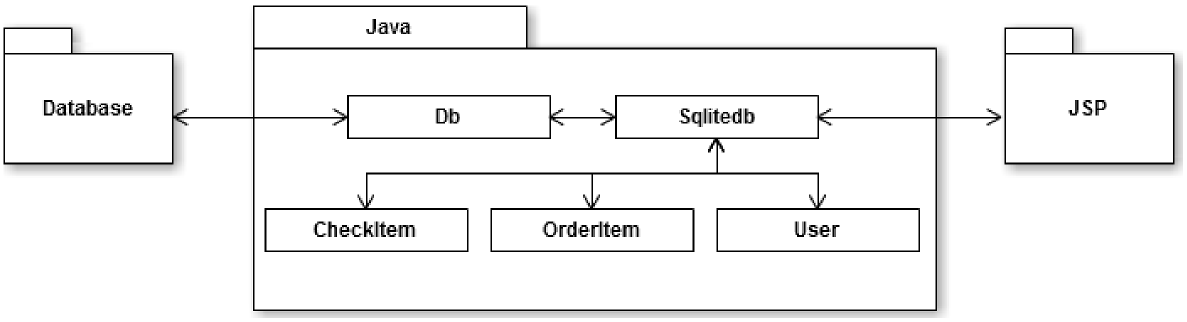


Figure 3. An overview of the classes in the Java part and the flow of information between them. All communication with the database is routed via Db.java while Sqlite.java handles the communication with the JSP part.

The files CheckItem.java, OrderItem.java and User.java are classes that define different kinds of objects. These are used by the methods of SqliteDb.java. Db.java has an exclusive connection with the database while Sqlitedb.java handles the operations required by the JSP part. The Java part thus acts as an intermediary, transferring data between the JSP files and the database.

These classes are described more in detail below.

4.1.1 Db.java

This is the class that contains all the methods needed for a connection with the database. It is exclusively SqliteDb.java that uses the methods of Db.java. This is also one of the classes that remain from the original project and has not been changed significantly.

4.1.2 SqliteDb.java

SqliteDb.java consists of several methods used for reading, writing, and updating of data concerning orders, employees and the check-list to the database.

This class is also the one that is used to establish a connection between the database and the program itself. It is one of the original files that remain but has been heavily modified during this project.
Below are the methods included in this class along with a description of their functionality.

getUsers()
Returns an array list of all employees.

getUsersByProf(String prof, String date)
Returns an array list with competence *prof* available on the date *date*.

insertUser(String telnr, String firstname, String lastname, String profession)
Adds an employee.

updateUser(String empl,String telnr, String firstname, String lastname, String profession)
Called when any of the variables within the above parenthesis needs to be changed for an employee.

deleteUser(int empl)
Removes an employee.

getUserInfo(int id)
Obtains information about an employee.

getProffesions()
Returns an array list of available competences within DISA.

getWorkOrders()
Returns an array list of all orders.

getWorkOrdersByDate(String date)
Gets orders for a certain date. Returns an array list.

addWorkOrder(String title, String desc, String date, String reqcomp, int prio, boolean done)
Creates a new order.

deleteWorkOrder(int id)
Removes an order.

updateWorkOrder(int id, String title, String desc, String date, String reqcomp, int prio, boolean done)
Is called when any of the variables within the above parenthesis needs to be changed for an order.

getWorkOrderInfo(int id)
Obtains information about an order.

insertLocking(int orderId,int employeeNbr, String date)
Is called when an employee is assigned to an order. Makes sure that he/she is added to the table of busy employees on the date in question.

getLocking(int id)
Returns an array list containing the employees bound to the given order as well as the date this order is due to be carried out.

deleteLocking(int id)
Deletes all the information stored for a given row. This makes the employees bound to this order available on the date when the order should have been carried out.

updateLocking(int id, int employeeId,String date)
Updates existing information.

4.1.3 CheckItem.java

CheckItem is a class for the objects in the check-list. It contains only a constructor and get and set methods for the different attributes of the CheckItem object.

4.1.4 OrderItem.java

A class used to create a new order object. Contains get and set methods to allow for the collection of information about said objects. An example of such information is the order number.

4.1.5 User.java

A class used to create employee objects. Contains get and set methods to allow for the collection of information on existing employees. An example of such information is the employee's identification number. This is also one of the original files and have not been modified significantly.

4.2 The JSP part

This part consists of several JSP files. These files can be divided into different groups. They make up the dynamic functions needed for MOS's functionality.

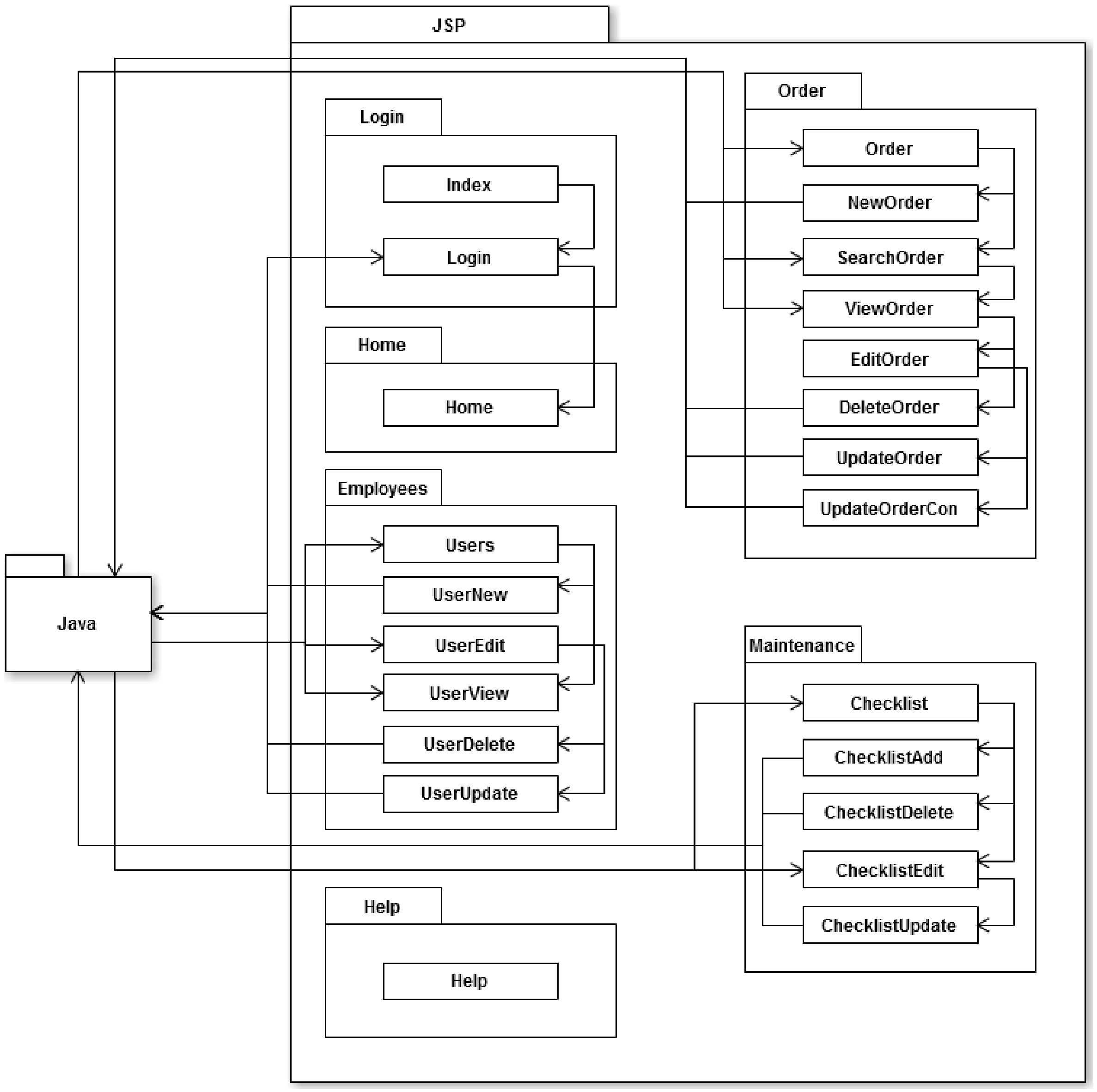


Figure 4. An overview of the JSP part. As can be seen, the different JSP files are clustered into logical groups by the purpose they serve. The arrows indicate the flow of information both between the individual JSP files and also between the files and the Java part.

The JSP part is used to send and retrieve data from the website. The JSP files can send data between themselves and to the Java part but not to the database directly. All communication with the database, whether it is to retrieve information or insert ditto into the database, is done via the methods of Sqlitedb.java. Sqlitedb.java then relays it to the database through the connection established by Db.java.

In our case the JSP files also consist of certain HTML code needed for the layout of the web page.

The groups, their files and a description of the functions in this project are listed below.

4.2.1 Login

To access the program you have to log in, which is what these JSP files handle. Logging in is done by entering a user name and a password. If these are found in the database and are correct the user will be redirected to the home page, else an error message will appear.

Index.jsp
This is the welcome screen and is automatically displayed when the program is opened. When clicking the ‘Login’ button the entered information is relayed to Login.jsp.

Login.jsp
Checks if the entered username exists in the database and if the password matches the one in the database. If everything is found and matches the user is relayed to Home.jsp, else an error message is displayed.

At the time of delivery there are two users added. Additional users can be added by entering them manually into the database.

Because of time constraints no regard for safety has been taken.

4.2.2 Home

If the correct username and password were entered the user is granted access to all pages and functions. By default the user is first redirected to the home page. Here the orders for today's date are displayed together with the option to make changes to the orders. Also the number of mouldings read on the machine can be entered.

4.2.3 Order

This group includes several files. What they are and what role they cover is described below.

Order.jsp
Allows the user to choose between creating a new order or search for an order.

SearchOrder.jsp
The user enters a date in a field so as to obtain all the orders to be executed at that date. Here, the user can also choose to edit any of the found orders.

NewOrder.jsp
Enables a user to add an order. Contains fields for entering the title, description of the order, the date when the order is to be executed, priority level, the necessary competence and selection of workers in selected skills. It also ensures that the information is stored in the database.

ViewOrder.jsp
The user is redirected here after pressing the ‘Edit’ button. This page shows all the relevant information regarding an order: title, description, date of execution, priority, selected skills, selected workers, if it has been carried out or not and order id number.

EditOrder.jsp
The user can modify the information contained in the selected order. The information here is the same as the information mentioned for ViewOrder.jsp.

UpdateOrder.jsp
This file makes sure that all of the data changed in EditOrder.jsp except selected users is updated in the database.

UpdateOrderCon.jsp
Ensures that selected users are updated in the database if they are changed in EditOrder.jsp

DeleteOrder.jsp
Deletes an order.

4.2.4 Employees

Below are the files that are covered in this group and their function in this program.

Users.jsp
Showing all employees with their full name and expertise. In front of each row is a small image with which one can access UserView.jsp by clicking it. Moreover, there is a button to add new employees.

UserNew.jsp
Allows the user to enter information about the employees who will be added. This information is the employee's first and last name, telephone number and competence.
This file ensure that this information is also entered into the database.

UserView.jsp
Displays the stored information about the selected employee. The information is the employee's ID number, full name, telephone number and competence.

UserEdit.jsp
Gives users the ability to change the stored information of a selected employee with the exception of the identification number. This information is the same as in the previous paragraph.

UserUpdate.jsp
Ensures that updated information in UserEdit.jsp is updated in the database.

UserDelete.jsp
Deletes the selected employee from the program.

4.2.5 Maintenance

This page contains the check-list where the service manager can view what operations have to be done at which intervals. This page also uses several files for its functions.

Checklist.jsp
This file retrieves the information of the check-list from the database and displays it when the ‘Maintenance’ tab is selected.

ChecklistAdd.jsp
The user can add new entries to the check-list. Shows a form in which the desired parameters can be entered and added to the database.

ChecklistDelete.jsp
Deletes the chosen entry from the database.

ChecklistEdit.jsp
Displays a form with the parameters of the chosen entry which can be edited and updated in the database.

ChecklistUpdate.jsp
Updates the chosen entry with the selected values in the database.

4.2.6 Help

This part of the program was mostly retained from the program created during the project in autumn 2011.

The idea is using this tab to get the necessary support when needed. This support is obtained by sending a questionnaire to DISA with a description of the problem, name and address of the person who submits the form. Alternatively you can call to DISA directly when there is contact information available on this page.

In addition to the questionnaire as well as contact information there is a user manual, service manual, and questions and answers which you can browse through before using the questionnaire or contact DISA in order to verify whether there is a solution to any problem in any of these three elements.

Unfortunately, these functions of this tab could not be implemented due to temporal restraints.

4.3 The database

The database is a SQLite database. It contains all of the information that is used by MOS. It is designed as follows:

Competences
Contains all of the professions possessed by the employees. The table *users* references it for its own column *profession*.

Checklist
Holds the information on the check-lists. Each post in the check-list is identified by its *checkId*. This table is currently stand alone and has no relations to the others in the database.

Login
Contains the unencrypted usernames and passwords. The username has to be unique and is used as the primary key of this table.

LastLogin
This table keeps track of which users have logged in previously and also when that occurred.

Users
Holds the information on the employees, such as their names and phone number. This table references the table *competences* for the column *profession*.

WorkOrder
This table contains the information on the work orders that are to be carried out and also the ones that are already done. It uses *orderId* as primary key as all orders have to be uniquely identifiable.

Locking
Locking is used to determine if a certain employee is available or not during a chosen day. It references both the tables *Users* and *WorkOrder*.

Mouldings

Keeps track of the number of mouldings done by the machine. Currently this information is not used and the table lacks connections to the rest of the tables. *Id* is not really needed but is in this case used to overwrite the same row as not to fill the database with useless information that will never be used or cleaned out.

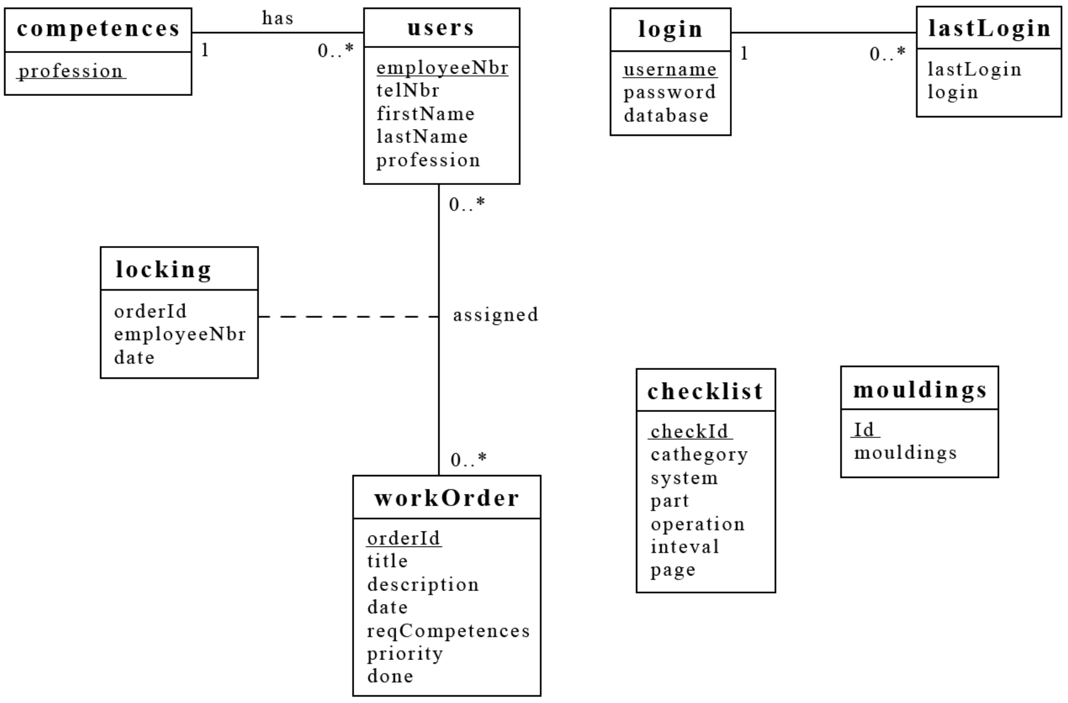


Figure 5. Diagram of the SQLite database used by MOS

4.4 Other elements

In addition to the three main parts there are some other smaller parts that are of importance to the project. Below is a description of these parts.

4.4.1 Style.css

The CSS file that holds the information concerning the programs graphical layout. There are declarations regarding what colour the background should have, which font to use, what colour the text should be, text placement and more.

4.4.2 Internationalization

At the time of delivery MOS will have support for both English and Swedish language.

For the language support the fmt tag library of the JavaServer Pages Standard Tag Library (JSTL) was used [17]. It provides among others functions for setting the locale of the user and for handling which language file to use.

The different languages are saved as external .properties files. These files have language codes in the file name so the program can choose the correct one. For example, in MOS the Swedish text file is called MessagesBundle_sv.properties. The 'sv' in the file name denotes that it is a Swedish file. For other languages the language code must be substituted for the appropriate one.

The language files use the same keys for the corresponding phrases and different translations. Two snippets of the language files are shown to illustrate this:

From the Swedish file:
```
index.welcome = Välkommen!
index.greeting = Var god mata in dina inloggningsuppgifter i fälten
till höger.
```

From the English file:

```
index.welcome = Welcome!
index.greeting = Please enter your username and password in the boxes to the right.
```

To use these phrases first JSTL has to be installed. In the case of MOS this was done by simply putting the file *jstl-1.2.jar* in the folder */WEB-INF/lib* .

At the top of each file that is meant to be internationalized search paths and other parameters for fmt have to be set.
In the case of MOS it looks like this:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<c:set var="language" value="${not empty param.language ? param.language : not empty language ? language : pageContext.request.locale}" scope="session" />
<fmt:setLocale value="${language}" />
<fmt:setBundle basename="i18n.MessagesBundle" />
```

To change the language file fmt is using the value of the ‘language’ variable has to be changed.

To use the different translations all output text is replaced by fmt tags with the corresponding key. For example, to write ‘Welcome!’ on the screen the following code is used:

```
<fmt:message key="index.welcome"/>
```

By using the fmt tag library for internationalization translation of the software becomes easy and requires minimal engineering since the .properties files are external and can be edited with most text editors.

4.4.3 Calendar

In order to give the user an overview of the orders over time there was a calendar implemented in MOS. The calendar can be seen in the areas where it was considered that time overview was important.

The code for this calendar was found on the Internet [18]. However, this code was modified so the current date is highlighted.

5 Results

This project resulted in a working program dubbed MOS. MOS is a program created to help DISA's customers keep track of when various machines and their parts need to be maintained.

Using MOS, DISA's customers can see what maintenance, referred to as work orders in the program, are to be carried out during the day. There is also the opportunity to see what should be done in the future. This is done by a search feature to seek orders for a given date. In addition, a user may add, remove and edit orders.

MOS also provides customers the ability to add, edit and delete the information of a particular employee.

In MOS, DISA's customers may read the check-list used to determine what maintenance work is needed. Users may also add, delete or edit rows of the default check-list to customise it for their special needs.

Should the user have any questions or concerns then it is possible to find contact information to the DISA offices and to DISA's employees in the ‘Help’ tab. This part is not quite fully implemented because of the limited time frame of the project.

The time frame of the project lead to that some of the requirements that were set up in the beginning of the project were not completed. Below is a description of the completed requirements as well as those which are yet to be implemented.

R1 Check-list

1. Ability to add entries
   Completed.
2. Ability to remove entries
   Completed.
3. Ability to edit entries
   Completed.
4. Ability to change time scope.
   Completed.

5. Ability to scroll through the check-list's pages using next/previous buttons.
   Not needed since the design is a scrollable, vertical list.
6. Contents of the checklist are inserted / updated in the calendar.
   This is part of the planned automatization which was deemed too time consuming for this project and is thus not yet implemented.

R2 Planning calendar

1. The calendar should show one month at a time.
   Completed.
2. Clicking on a day should reveal a more detailed picture of the tasks to be performed.
   Due to the lack of knowledge within this area as well as time constraints, this part was not implemented. However, a number of unfruitful attempts were made.
3. The calendar should show the data in different colours depending on the task priority.
   This requirement was not implemented due to prioritization (and time constraints) of the requirements with the largest impact on functionality
4. Adding other tasks apart from those present in the check-list should be possible.
   Completed.
5. Ability to reschedule a task.
   Completed.
6. Ability to remove a task.
   Completed.
7. Ability to add a time at which the task is to be performed.
   Completed.

R3 Resources

1. Ability to see available employees.
   Completed.
2. Ability to see available spare parts.
   Not yet implemented.

R4 Priority levels

1. There are three different levels: urgent, semi-urgent, not urgent.
   Completed.

2. The calendar represents every level with a colour; urgent - red, semi-urgent - orange, not urgent - yellow.
   This was not implemented but replaced by a simple numbering from one to three, where 1 = urgent, 2 = semi-urgent, 3 = not urgent. The numbers become visible when accessing the detailed view by clicking an order.

R5 Language support

1. The program should have support for multiple languages.
   Completed.
2. Upon delivery, the program should support the following languages: English and Swedish.
   Completed.
3. The language support should be easy to extend with additional languages without having to re-engineer the entire platform.
   Completed. This was accomplished with external language files that can be edited with any text editor. No engineering skills are needed for translating.

R6 Security

1. The program should be protected against SQL injections.
   Not implemented due to time constraints.

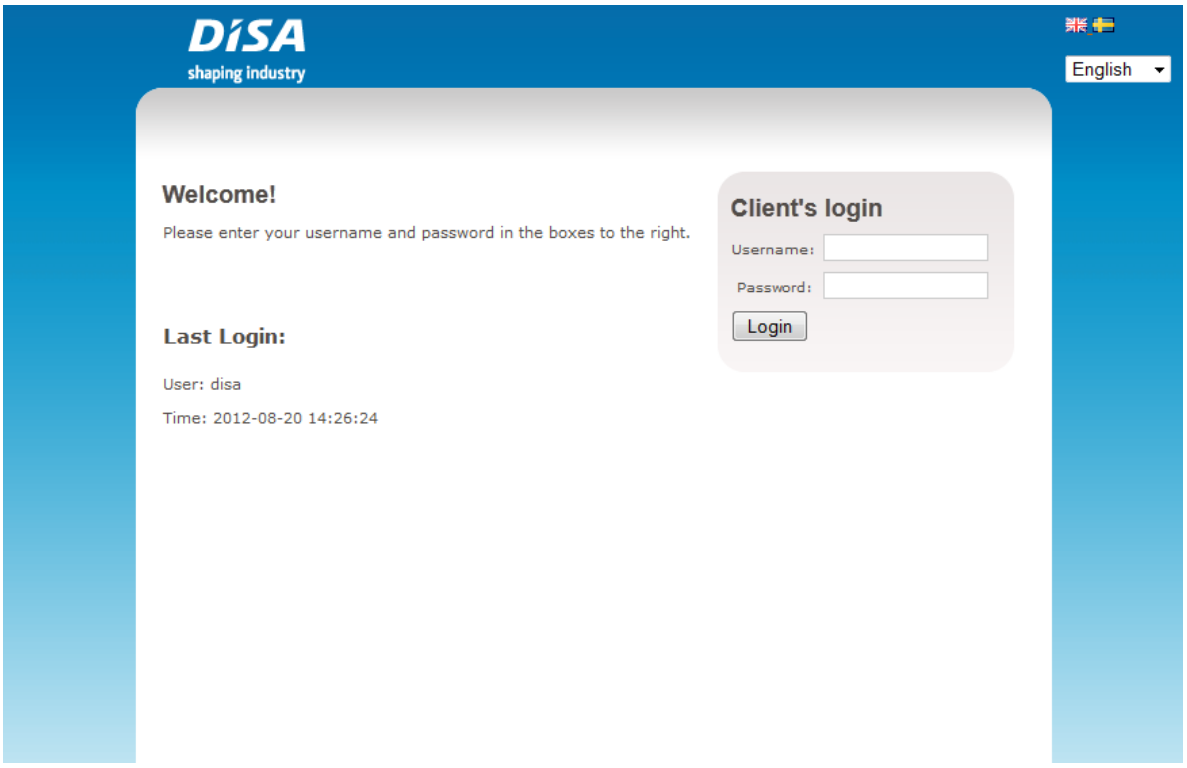
R7 Reporting

1. It should be possible to bring up a particular service report.
2. It should not be possible to edit a service report.

None of the requirements related to reporting were implemented due to time constraints. They were also considered not to enhance the functionality of MOS greatly at this point.

5.1 Screenshots

5.1.1 The login screen



Please enter your username and password in the boxes to the right.

User: disa

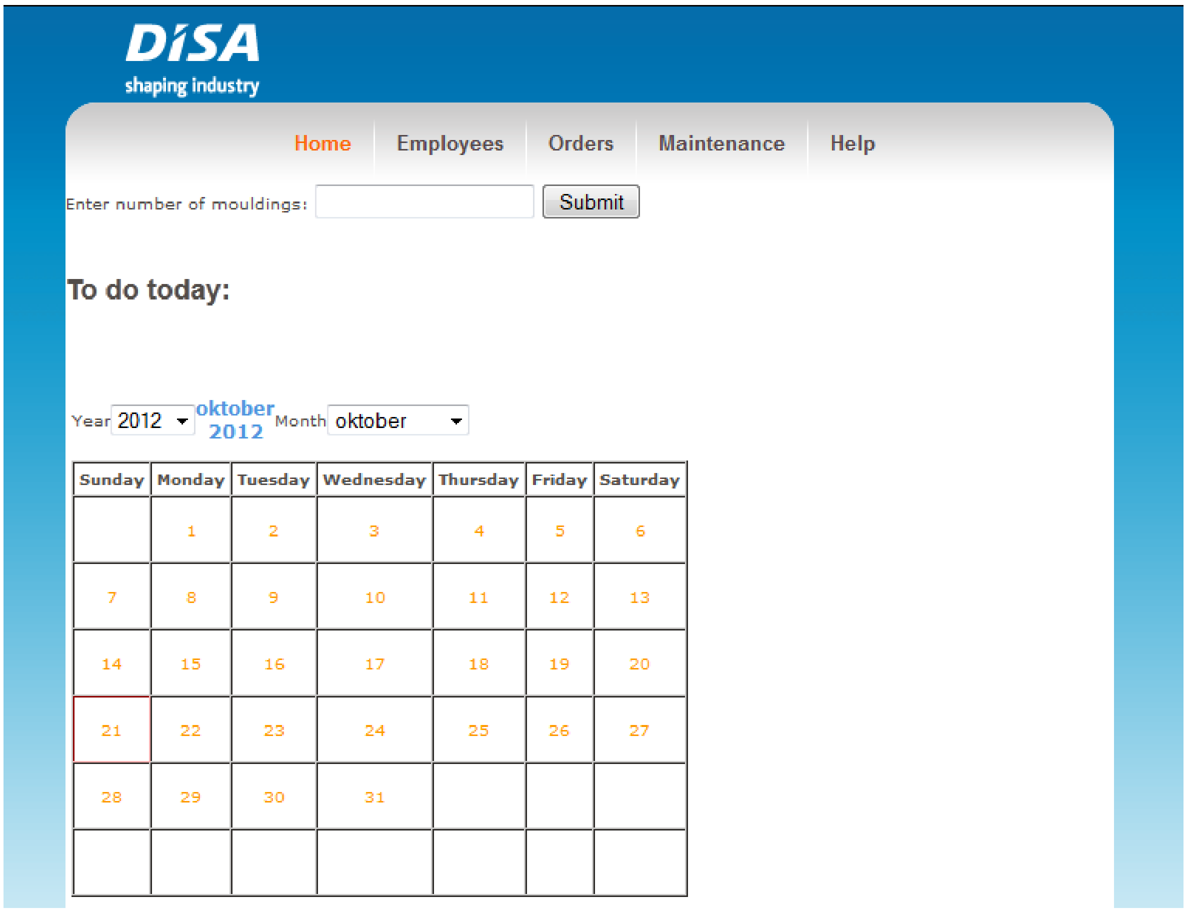
Time: 2012-08-20 14:26:24

Figure 6. The login screen. This is the first screen displayed when the program is opened.

When the program is opened the first screen that greets the user is the login screen, shown in figure 6. Here the user can enter his or her username and password to log in and also see who was last logged in and when. In the top right corner is the menu for choosing what language the page should be shown in. This is identical on all the pages.

When the user logs in he or she is sent to the home screen, shown in figure 7.

5.1.2 The home screen



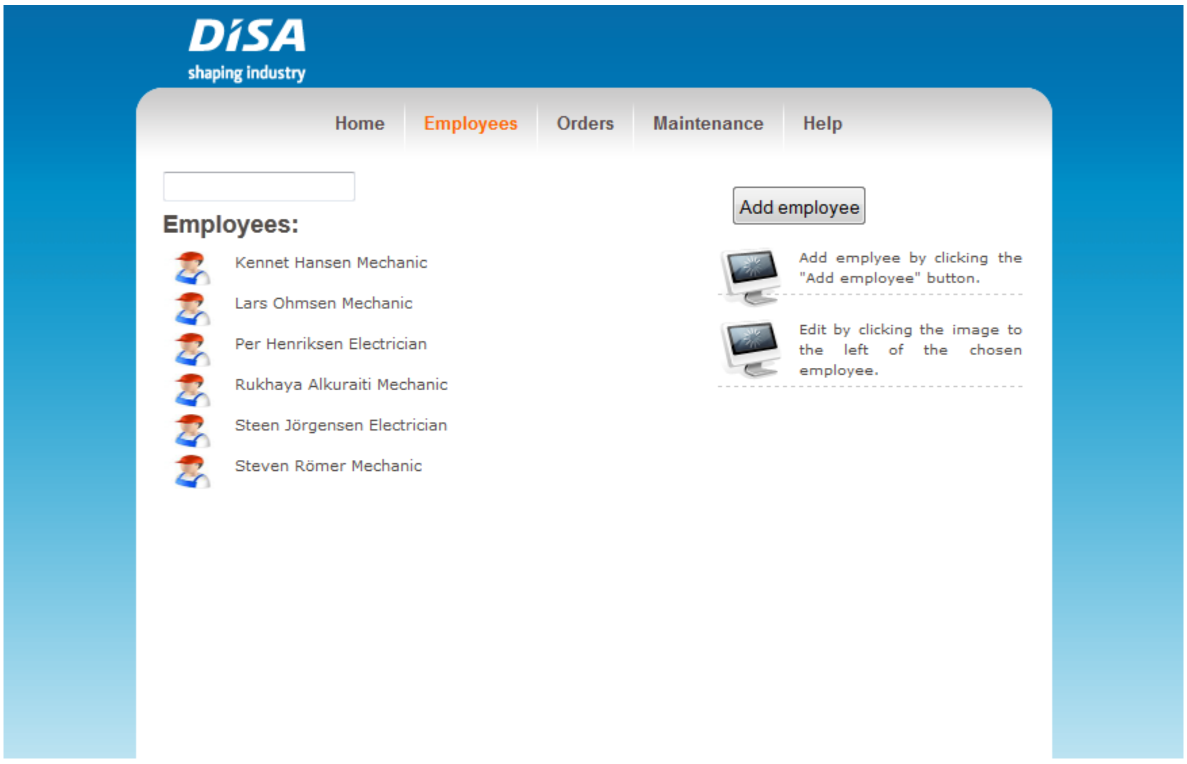
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Figure 7. The home screen.

The home screen offers the possibility for the user to enter the number of the mouldings done and also contains a calendar and a list of tasks that need to be taken care of that same day.

At the top of the screen is the menu for choosing which page to view next.

5.1.3 The employee screen



Employees:

Kennet Hansen Mechanic
Lars Ohmsen Mechanic
Per Henriksen Electrician
Rukhaya Alkuraiti Mechanic
Steen Jörgensen Electrician
Steven Römer Mechanic

Add emplyee by clicking the "Add employee" button.

Edit by clicking the image to the left of the chosen employee.

Figure 8. The employee screen.

The employee screen can be seen in figure 8. It contains a list of the currently registered employees and offers various options in handling the employee information.

5.1.4 The order screen

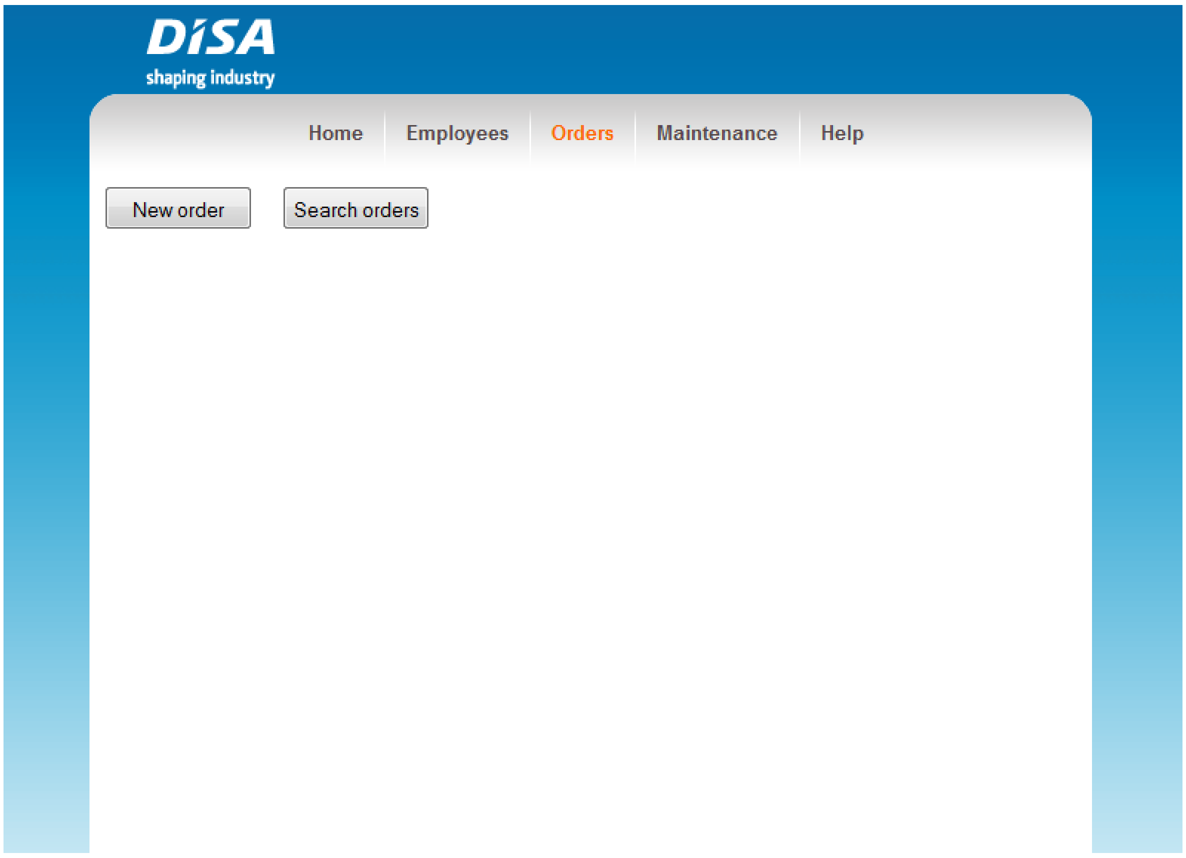
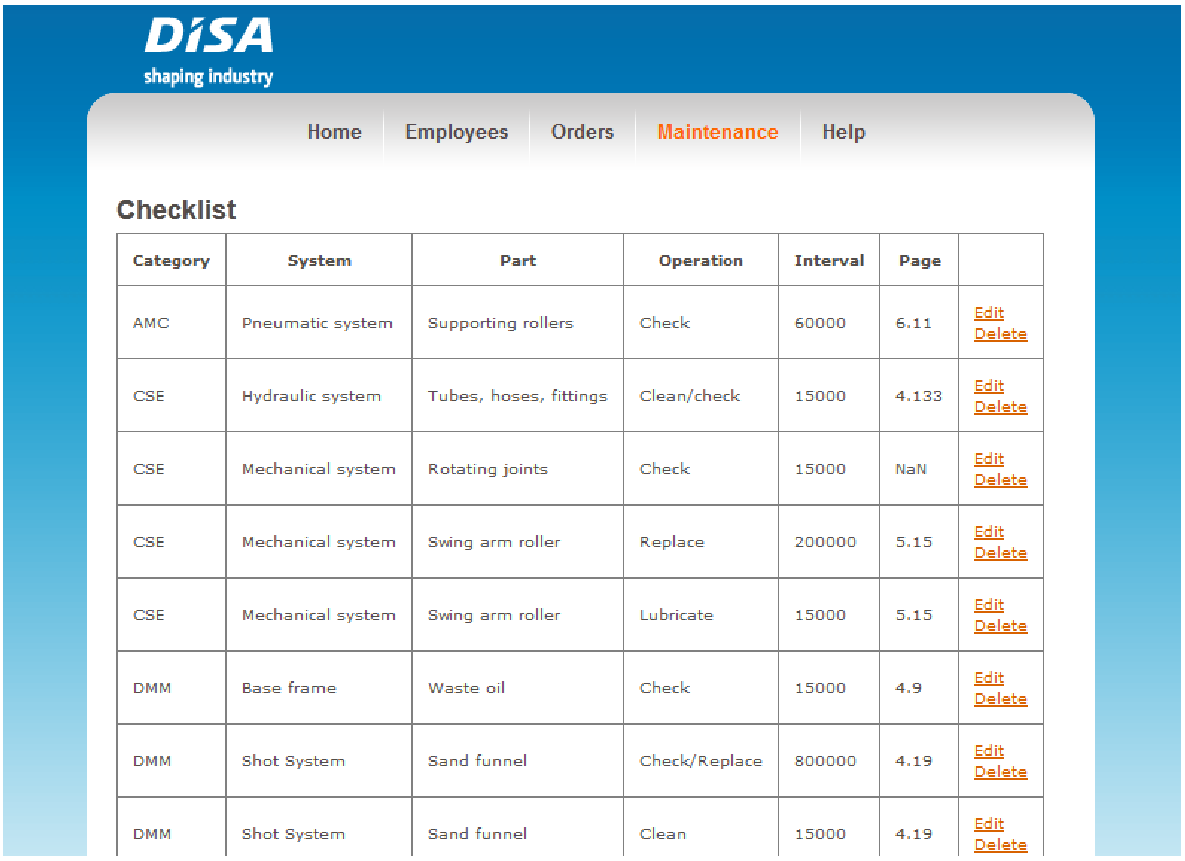


Figure 9. The order screen.

Figure 9 shows the order screen. At this screen the user can either add a new order or choose to search among the currently added orders.

5.1.5 The maintenance screen



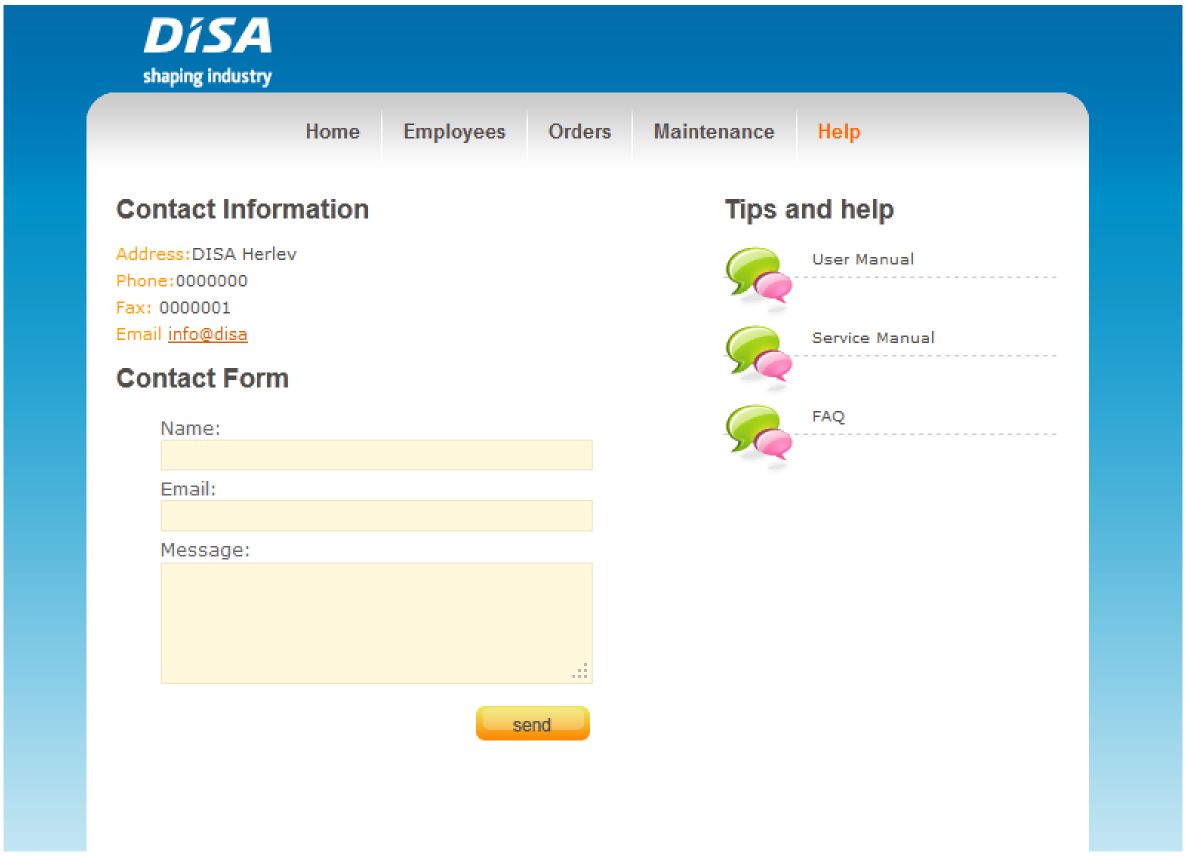
Checklist

Category	System	Part	Operation	Interval	Page	
AMC	Pneumatic system	Supporting rollers	Check	60000	6.11	Edit Delete
CSE	Hydraulic system	Tubes, hoses, fittings	Clean/check	15000	4.133	Edit Delete
CSE	Mechanical system	Rotating joints	Check	15000	NaN	Edit Delete
CSE	Mechanical system	Swing arm roller	Replace	200000	5.15	Edit Delete
CSE	Mechanical system	Swing arm roller	Lubricate	15000	5.15	Edit Delete
DMM	Base frame	Waste oil	Check	15000	4.9	Edit Delete
DMM	Shot System	Sand funnel	Check/Replace	800000	4.19	Edit Delete
DMM	Shot System	Sand funnel	Clean	15000	4.19	Edit Delete

Figure 10. The maintenance screen.

The maintenance screen shown in figure 10 displays the annual check-list. This list is vertically scrollable and entries can be added, edited or deleted to customize the check-list for the customer's special needs.

5.1.6 The help screen



Contact Information

Address:DISA Herlev
Phone:0000000
Fax: 0000001
Email info@disa

Contact Form

Name:

Email:

Message:

Tips and help

User Manual

Service Manual

FAQ

Figure 11. The help screen

The help screen (displayed in figure 11) offers assistance if the need should arise. Both a User Manual and a Service Manual is available and for further support there is a contact form and contact information to the user's local DISA office. No actual functionality, except for internationalization, is implemented on this page at the time of delivery. Currently this page serves the purpose of showing the possibilities.

6 Conclusions and comments

In the current situation DISA's customers themselves need to keep track of when service is needed for their equipment. The result of this is that the equipment does not last as long as expected when one forgets to perform service on time. DISA therefore decided to develop a platform that will help their customers keep track of service for their machines so as to ensure that the equipment lasts just as long as they should or even longer.

This lead to the development of the platform MOS. Expectations and hopes are that MOS will be able to act as an assistant to the customers as it helps to keep track of the maintenance.

As mentioned in the beginning of this report a meeting was held with DISA to clarify their requirements and preferences regarding MOS. Based on these requirements and requests MOS was implemented. When comparing the results and what DISA wanted one may say that this project was a successful one.

MOS currently provides DISA's customers with the ability to add, delete and edit orders and employee information. In addition, the user of this program can see the Annual Checklist and edit it. By edit, we mean to add, delete or update posts. With MOS, the user may also view his/her work orders, see when they are due to be executed and also see an overview of his/her employees.

After meeting with DISA we had several ideas on how we could develop MOS and what we felt that MOS should offer. Unfortunately, because of the time frame of the project we had to exclude some of these ideas.

Just like everything in software development, MOS can be further developed to become even more efficient and useful than it is when delivered.

Below are examples of components that can be developed or added to improve MOS. These parts are also the additional ideas that we had to exclude.

6.1 Future development

As mentioned in this report, MOS includes a calendar application. At the start of the project the idea was that this calendar would be clickable. You would click on a particular day and thus get all the orders for that day. Since this is not implemented in this version of the program it may be something to include in future versions.

The security issue has been mentioned several times in this thesis but should be included in this topic too. There is quite a lot to improve on the security since at the time of delivery it is seriously lacking. Both password protection and protection against SQL injections should be taken into consideration.

The current version of the MOS takes into account the priority of orders. What may be appropriate to implement in future versions is that orders may be tinted different colour depending on their priorities. In this way the user can directly get an overview of how the orders are prioritized. In the present situation the order is prioritized with a number that is visible when you edit an order.

Another idea was that the functions of MOS would be handled automatically. MOS would manage everything related to when service should be done, what task should be carried out, removing an order when its time has passed and when its absolute deadline is. However, this idea was excluded rather quickly as it was considered to be far too time consuming.

7 Dictionary

ACID - **A**tomicity, **C**onsistency, **I**solation, **D**urability, a property set for database transactions the ensure reliability

Apache Tomcat – An open source web server developed by the Apache Software Foundation

DISA – International company producing metal casting technology

Internationalization - The process of designing a program so that it can be translated and adapted for various regions without engineering changes

JSP - JavaServer Pages

JSTL - JavaServer Pages Standard Tag Library, a Java EE Web application development platform component that adds tag libraries to JSP

MOS – Managed Operations Software

SQLite - A relational database management system

8 References

Most of the information for this thesis has been obtained from the official web sites of projects and their developers since they are considered to be the most reliable. Reference 19 was used as HTML reference during the project and is thus not mentioned in the text.

1. http://www.disagroup.com/en/sites/disa/content/disa_home.aspx (March 2012)
   Official web site of DISA
2. http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/09-JSP-Intro.pdf (August 2012)
   An introduction to JSP
3. http://java.sun.com/developer/onlineTraining/JSPIntro/contents.html (August 2012)
   Online JSP course by Sun
4. http://webmaster.gsfc.nasa.gov/presentations/jsp.pdf (August 2012)
   A presentation on JSP by the GSFC Webmasters
5. http://www.php.net/ (August 2012)
   The official web site of PHP
6. http://seancoates.com/blogs/security-and-driving-and-hiring (August 2012)
   Blog entry on the security of PHP
7. http://phpids.org (August 2012)
   Official web site for PHPIDS
8. http://mashable.com/2010/11/19/pros-cons-php/ (August 2012)
   ‘8 Experts Break Down the Pros and Cons of Coding With PHP’
9. http://www.ehow.com/info_8696951_compare-mysql-vs-sqlite.html (April 2012)
   The web page used for comparison of MySQL and SQLite
10. http://www.sqlite.org/ (March 2012)
    Official SQLite web site
11. http://www.sqlite.org/transactional.html (March 2012)
    Explanation on ACID
12. http://tomcat.apache.org/ (March 2012)
    Official web site of Apache Tomcat

13. Jason Brittain; Ian F. Darwin (2003)
Tomcat: The Definitive Guide
O'Reilly Books. ISBN 0-596-00318-8
14. http://www.dropbox.com/ (March 2012)
Official web site of Dropbox
15. http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junor (March 2012)
Official download site for Eclipse EE
16. http://code.google.com/p/sqlite-manager/ (April 2012)
Official web site for SQLite Manager extension for Firefox
17. http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/ (June 2012)
Documentation on the JSTL fmt tag library
18. http://www.easywayserver.com/blog/jsp-calendar/ (April 2012)
Web page where the calendar was obtained
19. http://www.w3schools.com/html/default.asp
Used as reference for HTML

8.1 Image source

Figure 1: http://www.disagroup.com/en/sites/disa/content/disa_home.aspx
Official web site of DISA