Order Book and Volume Tracking Tool

Andreas Östergårds

Bachelor's Thesis

Information Technology

Vaasa 2012

BACHELOR'S THESIS

Author:	Andreas Östergårds
Degree Programme:	Information Technology, Vaasa
Supervisor:	Susanne Österholm

Title: *Order Book and Volume Tracking tool (OBVTT)*

Date 30.4.2012 Number of pages 37

Abstract

The thesis was done at the request of Wärtsilä Industrial Operations (WIO), a division within the Wärtsilä Corporation. The task was to create an application through which employees of WIO would be able via intranet to report delivery and order book data for different products, as well as track previously stored data and have it presented to them in a graphical format. The work consisted of planning and programming of database and application, while creating a stylish and user friendly interface. An SQL Server database was used for data storage, while the actual application was programmed in C# and HTML using the ASP.NET platform.

Language: English Key words: order book, volume tracking, reporting, ASP.NET, SQL Server

EXAMENSARBETE

Författare:	Andreas Östergårds
Utbildningsprogram och ort:	Informationsteknik, Vasa
Handledare:	Susanne Österholm

Titel: *Orderbok- och volymuppföljningsverktyg*

Datum 30.4.2012 Sidantal 37

Abstrakt

Examensarbetet utfördes på begäran av Wärtsilä Industrial Operations (WIO), en avdelning inom Wärtsilä Abp. Uppgiften var att skapa ett program där anställda inom WIO skulle, via intranätet, rapportera leverans- och orderbokdata för olika produkter, samt följa upp tidigare lagrad data och få denna presenterad i grafiskt format. Arbetet bestod av planering och programmering av databas och applikation, samt att skapa ett snyggt och användarvänligt gränssnitt. En SQL Server-databas användes för lagring av information, medan själva applikationen programmerades i C# och HTML under ASP.NET-plattformen.

Språk: engelska Nyckelord: orderbok, volymuppföljning, rapportering, ASP.NET, SQL Server

Förvaras: Theseus.fi och Tritonia, Vasa vetenskapliga bibliotek

OPINNÄYTETYÖ

Tekijä:	Andreas Östergårds
Koulutusohjelma ja paikkakunta:	Tietotekniikka, Vaasa
Ohjaaja:	Susanne Österholm

Nimike: *Tilauskirja- ja määräseurantatyökalu*

Päivämäärä 30.4.2012 Sivumäärä 37

Tiivistelmä

Opinnäytetyö annettiin tehtäväksi Wärtsilä Industrial Operationsilta (WIO), joka on osasto Wärtsilä Oyj:ssä. Tehtävänä oli luoda ohjelma, jossa WIO:n henkilökunta voisi intranetin kautta ilmoittaa eri tuotteiden toimitus- ja tilauskirjanpitotietoja sekä seurata aikaisemmin arkistoituja tietoja ja saada ne esiteltyä graafisessa formaatissa. Työhön sisältyi tietokannan ja applikaation suunnittelu ja ohjelmointi sekä tyylitellyn ja helppokäyttöisen käyttöliittymän luominen. SQL Server-tietokantaa käytettiin tiedon varastoimiseen, kun itse applikaatio ohjelmointiin C#:ssa ja HTML:ssä ASP.NET-alustaa käyttäen.

Kieli: englanti Avainsanat: tilauskirja, määräseuranta, tiedottaminen, ASP.NET, SQL Server

Contents

1 Introduction

1.1 About the employer

Wärtsilä, founded in 1834, is a Finnish based company specialized in power solutions for the marine and energy markets. Its main categories are ship power and power plants and its product lines include engines, automation equipment, propulsion equipment and energy efficiency products. The company employs approximately 17,500 employees worldwide and runs its operations in 160 locations in 70 different countries. Its headquarters are located in Helsinki. [1]

Fig 1. Wärtsilä headquarters in Helsinki. [2]

The actual employer, Wärtsilä Industrial Operations (WIO), is a division within Wärtsilä comparable to the sales division or service division. WIO serves Ship Power, Power Plants, services and licensees, covering the needs of operators, utilities, investors and industry. WIO interfaces directly with suppliers, licensees and research and development partners as well as external customers. Although WIO operates worldwide, the place of work was in Vaasa. [3]

1.2 The task

The “Order Book and Volume Tracking Tool”, or the OBVTT for short, is supposed to be a tool designed to give employees of Wärtsilä Industrial Operations an easy and accessible way to report and overview delivery and order book data at their respective factories.

Deliveries are defined as actual amounts of a product manufactured and delivered by Wärtsilä, while orders represent future deliveries. Slot reservations are reserved orders that can be considered probable and may change before actual orders are placed. The application should thus consist of two main functionalities: a tool for reporting deliveries, orders and slot reservations as well as a tracking tool that generates graphical data and prognoses based on previously stored information.

In addition to the main functionalities there should be administrative tools available for administrators, such as product editing and account management. In addition to the ASP.NET based application, an SQL database located on the company server is to be used for data storage.

Each singular employee should be given an account to which he or she logs on and reports data. Different credentials will be assigned to each user, limiting users to certain product centers, factories or products, depending on what data each person should have access to. Credentials are to be managed by administrators.

1.3 Requirements

The main requirement of the project is to create a tool that provides the necessary functionalities. Necessary functionalities include the ability to read and write product and order book data and to track this data over time. Once the required items are implemented and fully functional, there are several optional items that may be implemented in the future.

A list of requirements:

- The ability to store product information in the database
- The ability to store delivery and order book data for products
- The ability to view and track previously stored delivery and order book data
- The ability to administrate database content, e.g. accounts, factories and products
- A stylish yet simple graphical user interface that is easy to use
- A graphical design that stays true to the Wärtsilä color schemes and templates
- Guaranteed functionality

2 Development tools

2.1 Database

2.1.1 Type and server (Microsoft SQL Server 2008)

Microsoft SQL Server 2008 is a relational database server whose primary function is to read and write data as requested by other software applications running either on the same computer or on other computers across a network. It was initially developed by Sybase before being sold to Microsoft, becoming Microsoft's entry to the enterprise-level database market. The first release dates back to 1989, while the latest release is SQL Server 2012 released in 2012. [4]

Since the product involves reading and writing of data, a database would naturally be a critical element of the project. Wärtsilä operates its own internal SQL server, on which many of their in-house applications run. Since the database of the application could run on this server, using an SQL database was the most optimal choice.

During the development process a local SQL Server Express was used on the development computer. The Express edition is a limited free version of the SQL Server range of products, providing enough resources for the development of this product. The tool used to access and manage the database was SQL Server Management Studio, an application available as a part of the Microsoft SQL Server package. [5]

2.1.2 Query language (Transact-SQL)

Structured Query Language (SQL) is a cross-platform query language used for managing data in a Relational Database Management System (RDBMS), designed by Donald D. Chamberlin and Raymond F. Boyce and first introduced in 1974. [6]

Transact-SQL is an extension of SQL. It expands SQL by including features such as procedural programming, local variables and try/catch logic and is central to the use of Microsoft SQL Server. [7]

To interact with tables within a database, a query language is needed. A query is a command line that, when executed, may read or write data to or from the database. An example of some simple query commands can be seen in *Code example 1*. Since Microsoft SQL Server was to be used for the project, Transact-SQL automatically became the query language to be used.

Code example 1. Using SQL query code to create a table, insert two rows into it and then display the contents of the table.

```
CREATE TABLE ExampleTable
(
      name  VARCHAR(20) PRIMARY KEY,
      age   INT
)

INSERT INTO ExampleTable VALUES ('John Doe', 33);
INSERT INTO ExampleTable VALUES ('Jane Doe', 27);

SELECT * FROM ExampleTable;
```

2.2 Application

For the programming of the application, Microsoft Visual Studio 2010 was used. Visual Studio is an Integrated Development Environment (IDE) developed by Microsoft that supports several programming languages such as C#, C++ and VB, and is used to develop different applications for platforms such as Windows, Windows Phone and the web. [8]

2.2.1 Application framework (ASP.NET)

ASP.NET is a web proprietary application framework developed by Microsoft and first introduced in January 2002. It allows developers to build dynamic web sites and web applications combining .NET languages with traditional HTML and CSS. Its latest release is ASP.NET 4.0, released in 2010. [9]

There were discussions about which platform to use for the graphical user interface. First of all there was the choice between using a universal web based application that could be accessed from anywhere within the Wärtsilä network and a stand-alone client that could be installed on an employee’s laptop or desktop computer. Due to the inconveniences of keeping a stand-alone client up to date on every machine within the company, it was decided that a web application would be the most practical solution, since there would be only the one application running on the Wärtsilä server. Secondly, there was the choice of platform and programming language. Although PHP initially seemed like a logical choice, ASP.NET (using C#) was later considered the best alternative as previous applications developed for Wärtsilä had been based on the ASP.NET platform, which allowed analysis of the source code of other applications during the development process. The only downside to using ASP.NET instead of PHP was developer experience.

2.2.2 Server side programming language (C#)

C#, pronounced *C sharp*, is a cross-platform multi-paradigm programming language developed by Microsoft. Influenced by programming languages such as C++ and Java, it was first widely distributed in 2000 and is intended to be a simple yet modern programming language. The most recent version is C# 4.0, which was released on April 12th 2010. Using C#, developers may create software for platforms such as Windows, Xbox 360 and Windows Phone and the web. [10]

While it had been decided that ASP.NET would be used, there was the choice between using C# and Visual Basic (VB) as the main programming language. Simply because of developer experience, C# was considered a better alternative. VB is also quite different from C based programming languages in the way it is written and would likely have required some time to fully learn. *Code example 2* displays some simple C# code.

Code example 2. A tiny C# console application consisting of a string that is written to the command line.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ExampleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            string exampleString;

            exampleString = "A little example";

            Console.WriteLine(exampleString);
        }
    }
}
```

2.2.3 Markup language (HTML)

HTML, short for *Hypertext Markup Language*, is the main markup language used for web pages. The first version of the HTML standard was released in 1995, while the most recent version (HTML5) was released in 2008.

Elements used in written HTML are the basic building-blocks of a webpage and consist of tags enclosed within angle brackets as can be seen in *Code example 3*. Web browsers are designed to read and visualize web pages by interpreting the written HTML tags, generating the graphical webpage. [11]

Code example 3. HTML code rendering a webpage with the title “Example” displaying a small piece of text.

```html
<html>
<head>
    <title>Example</title>
</head>
<body>
    <div>
        A tiny example.
    </div>
</body>
</html>
```

In addition to the regular HTML elements, ASP.NET offers plenty of its own unique components. These components look similar to regular HTML elements, as can be seen in *Code example 4*, and also generate regular HTML code once executed on the server. Using ASP.NET components saves the developer plenty of time and work, the ASP.NET Calendar component being a good example. Creating a calendar from scratch using regular HTML along with some server-side programming language is quite a time consuming and difficult task, compared to using the ASP.NET component which lets you easily set up properties while the hard work is done behind the scenes. [12]

Code example 4. ASP.NET components mixed into regular HTML code, rendering a div containing some text and a button.

```
<asp:Panel runat="server">
    <h1>Example</h1>
    <p>
        This is an HTML paragraph within an ASP.NET panel
    </p>
    <br />
    <br />
    <asp:Button ID="Button" runat="server" OnClick="ButtonEvent" />
</asp:Panel>
```

2.2.4 Style sheet language (CSS)

CSS, short for *Cascading Style Sheets*, is a style sheet language primarily designed to separate presentation, such as layout, colors or fonts, from the content of e.g. an HTML document. By keeping presentation elements separate from content, shared formatting is enabled and content accessibility is improved. An example of a CSS class can be seen in *Code example 5*. [13]

Code example 5. A small CSS class that can be applied to any div element in an HTLM document, turning the div into a red square of 100 by 100 pixels.

```
div.css_example
{
    width: 100px;
    height: 100px;

    background-color: Red;
}
```

To create a visually satisfying user interface, most HTML and ASP.NET elements were associated with some CSS class implemented within a separate CSS file. One of the main problems when designing a web page or a web application is keeping the final appearance as identical as possible on all major web browsers. In an attempt to neutralize the different anomalies and offsets rendered by some browsers, a separate CSS file was used to reset all HTML elements to the same values, by assigning a default value to each element type. Reset files are commonplace in web design and can easily be found online.

2.3 Other tools and solutions

2.3.1 Graphics and design (Paint Shop Pro X2)

Paint Shop Pro, originally developed by Jasc Software, is a series of advanced raster and vector editing software available for PC and Mac, comparable to software such as the popular Photoshop series of editing software. The current X series of PSP is being developed by Corel, to which PSP was sold in 2004. Corel is known for other famous imaging products such as Corel Draw and Painter. [14]

For graphical design that could not be rendered within the solution simply by using CSS code, Paint Shop Pro X2 was used. Such graphics include pictures, icons and other special imagery. Certain documentation would also contain graphics created using PSP.

2.3.2 Tools and applications used for testing

Apart from using actual browser applications when testing the application, Microsoft Expression Web SuperPreview was used. SuperPreview is an application that is included as a part of Expression Web, which is one of the products available in the Microsoft Expression range of products. SuperPreview allows the developer to try out his or her web page or application on a range of different web browsers simultaneously, comparing the results side by side. This removes the need to install every single browser just for testing. An example of a SuperPreview comparison can be seen in *Figure 2*.

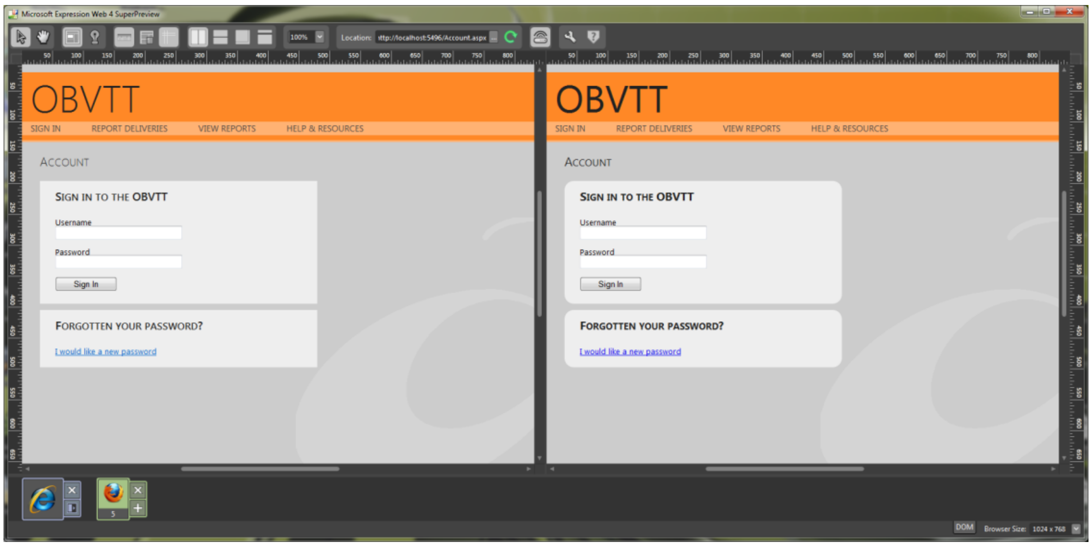


Fig 2. Comparing the OBVTT between Internet Explorer and Firefox using SuperPreview.

3 Concept

3.1 Project milestones

Milestones were set at the beginning of the project to give people involved a very basic idea of how the project would progress through different phases. The first phase, the “Concept” phase, would involve meetings and brainstorming, choosing the proper tools and solutions to use and so on. The “Prototype” phase would be somewhat of a middle phase, where the project was not yet finished but had reached a point where one could present its main design and functionality. The final “Product” phase would be reached once the product had met its requirements and undergone enough successful testing to be considered complete. Beyond the “Product” phase there would be the opportunity for improvements, updates and modifications as seen fit by the employer.

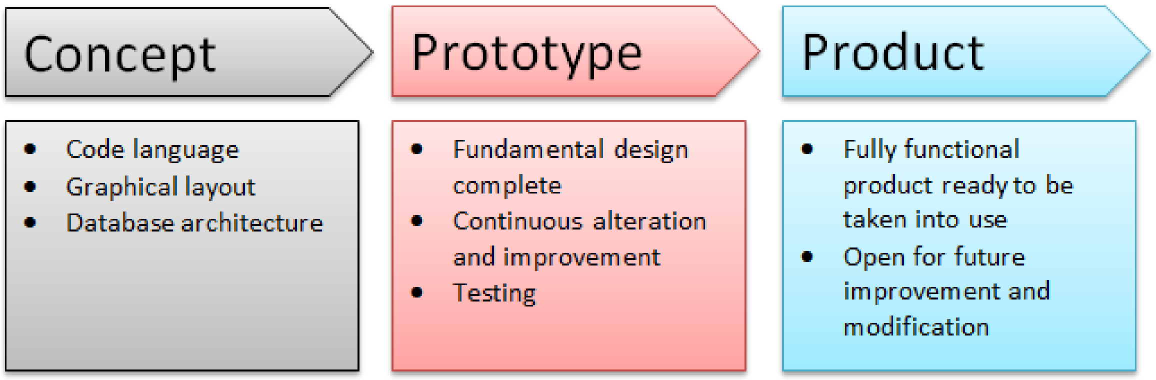

Concept
Code language
Graphical layout
Database architecture
Prototype
Fundamental design complete
Continuous alteration and improvement
Testing
Product
Fully functional product ready to be taken into use
Open for future improvement and modification

Fig 3. Project milestones.

Initially there were discussions about whether to set target dates for each milestone, but due to different inconveniences it was considered best to let the development progress at its own pace.

3.2 Understanding the production hierarchy

In order to fully understand how to design the database architecture, a visualized hierarchy image was created (*Figure 4*). The production within the company is structured into three main categories. The product center, which covers a certain set of factories, is the main category with factories serving as the secondary category. The third category is the product, which is covered by one or more factories. Credentials are based on these main categories throughout the application.

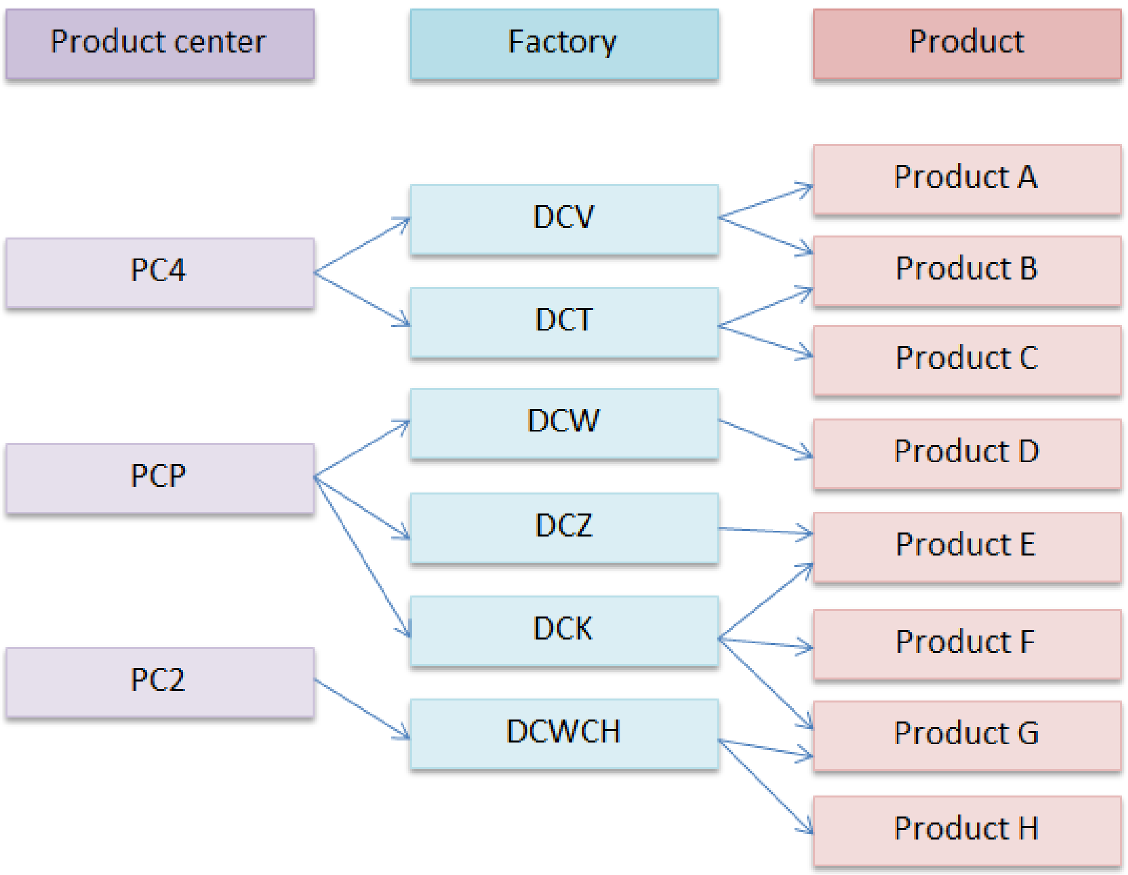


Fig 4. Wärtsilä product center hierarchy.

3.3 Application architecture

A block architecture image (*Figure 5*) was created to give a very simple but descriptive overview of the product. The image displays the three most integral parts of the application interaction: the user who interacts with the application, the framework (ASP.NET) on which the graphical user interface and server side application code is based and the database server (Microsoft SQL Server) where all the report data is stored in a database.

The user accesses the GUI, where he or she may access the main tools. These tools then interact with the SQL server database, reading and writing data. Finally the user may be presented with graphical data or messages, depending on which tools were used, completing the interaction.

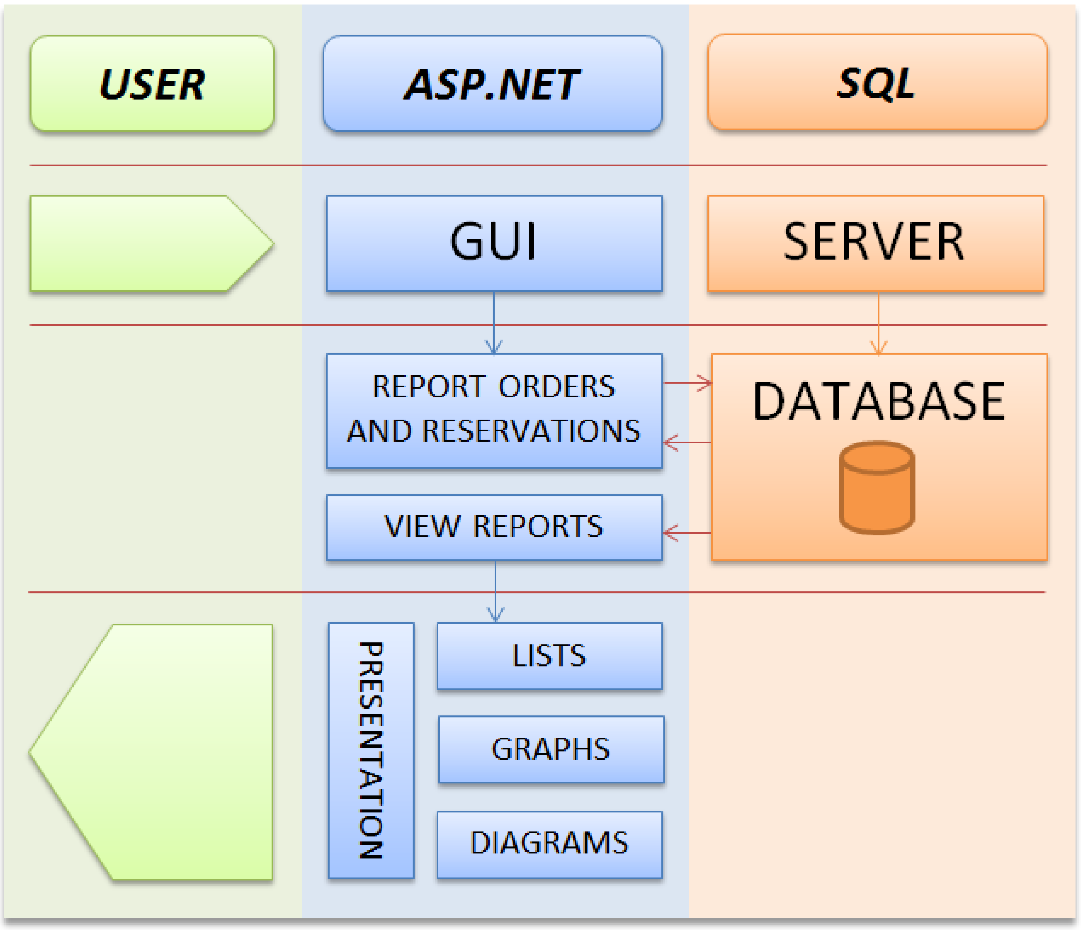


Fig 5. A simple visualization of the application architecture.

3.4 User interaction

A flowchart (*Figure 6)* was created to give a more detailed overview of the interactions of an individual user. The flowchart demonstrates a user signing in to the application using his username and password. The user's credentials are then validated before he or she is signed in as one of several user types. Depending on user type and credentials the user may view and/or edit certain factories. Being allowed to view a factory means that the user may fetch historical data for products belonging to said factory, while being allowed to edit a factory means that the user may report data for products belonging to said factory.

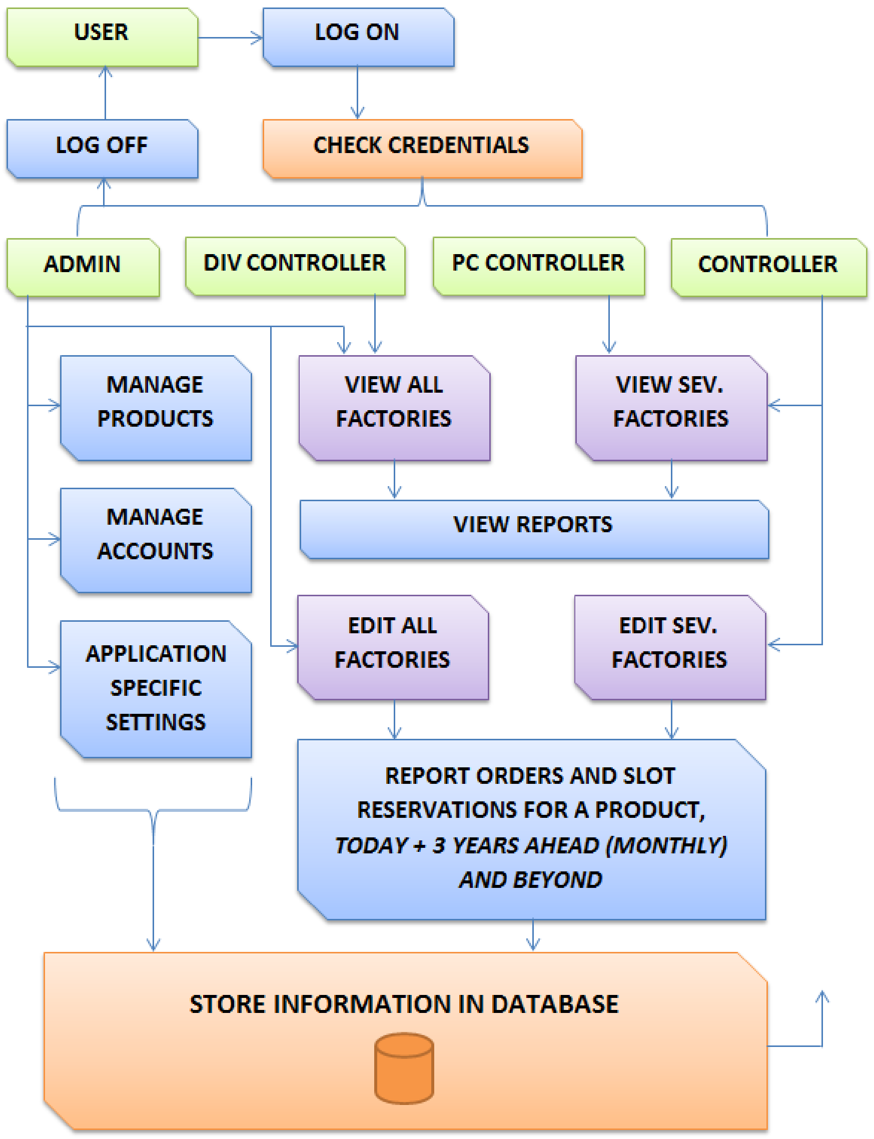


Fig 6. Application flowchart.

4 Development

4.1 Acquiring software

Most software used during the development process was available through the Academic Alliance, a web platform maintained by Microsoft from which students belonging to a member university may freely download software for educational purposes, while others may use it as an online store to purchase Microsoft products. Other software used during the development process had either been previously purchased or downloaded under freeware or trial licenses.

4.2 Database

4.2.1 Architecture

For an application like the OBVTT, the database could be considered the foundation of the entire project. All actions performed from the graphical user interface interact with the database in one way or another, thus the application would be rendered useless without a stable and logical database. Because of this, a great amount of time and planning was put into creating a database architecture that would certainly cover all requirements while also being left relatively dynamic, should any alteration be required during or after the development work.

Several prototypes of a database were created before the final architecture could be established (*Figure 7*). The final database architecture is very much based on the items covered in the “Concept” chapter, having tables reflect the Wärtsilä production hierarchy, credential management and delivery reporting.

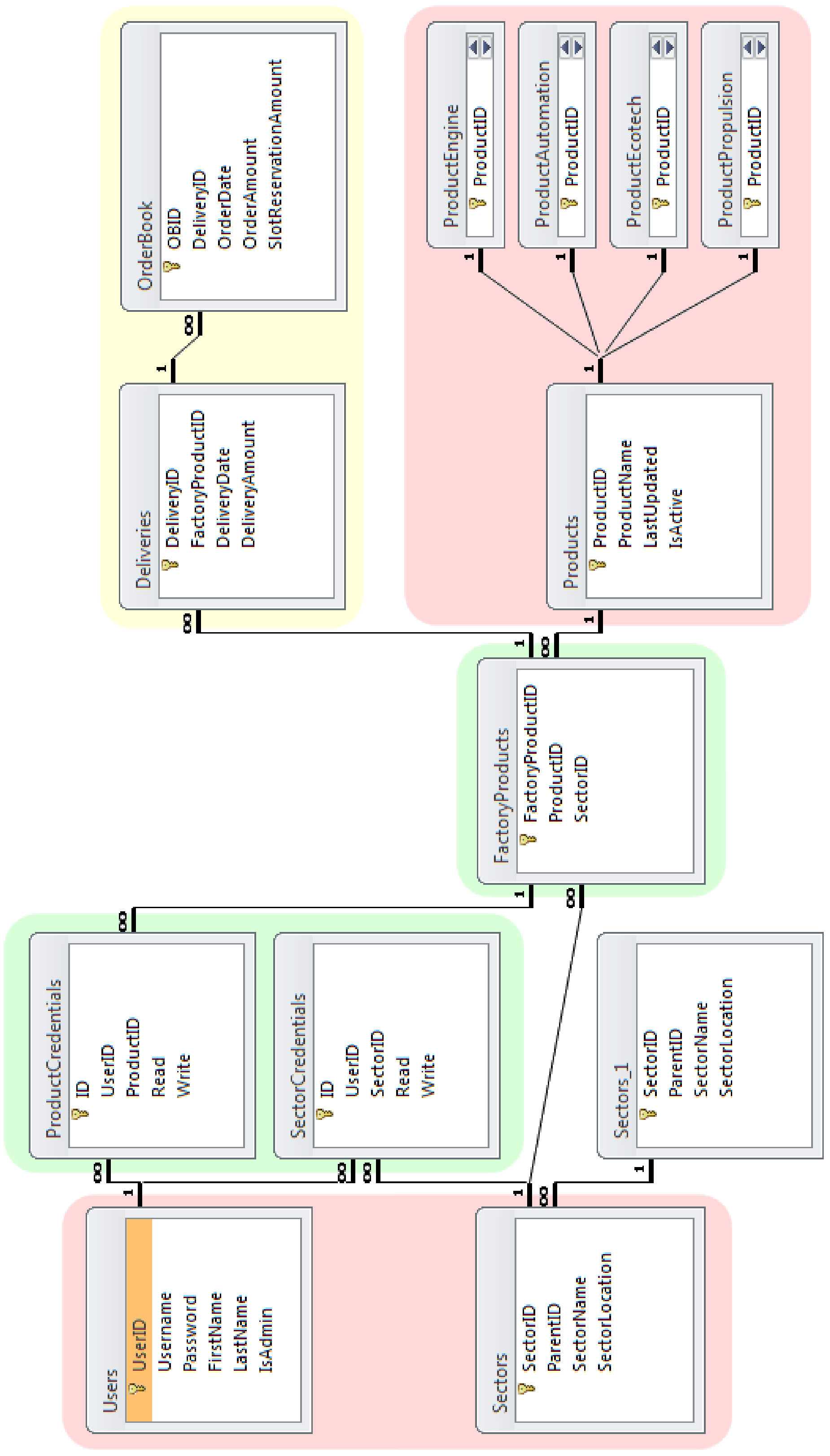


Fig 7. OBVTT database architecture. The red areas contain base data tables only editable by administrators, the green areas contain tables providing bonds between other tables and the yellow area contains the delivery and order book data tables editable by users. The table "Sectors_1" represents the self-reference between the SectorID and ParentID values of the "Sectors" table.

4.2.2 Tables in detail

The database consists of several categories of tables: one category reflecting the actors and objects used in the application (*Users, Sectors and Products*), one category maintaining credentials and bonds between previously mentioned actors and objects (*SectorCredentials, ProductCredentials and FactoryProducts*), one category consisting of different product types and configurations (*Engine, Automation, Propulsion, Ecotech*) and finally the *Deliveries* and *OrderBook* tables where the delivery and order book data is stored.

The *Users* table naturally stores user information, such as a username and password for signing in as well as some personal information. It also stores information about the user's administrative rights and whether the account is still active.

The *Sectors* table stores the product centers and factories used within the Wärtsilä production hierarchy. Since the attributes of these two objects are so similar, it was decided ideal to use only one table, pointing factories to product centers by using the "ParentID" column, which references to a "SectorID" within the same table. This ultimately means that a row missing a "ParentID" value is a product center, while the rest are factories. The advantage of using only one table is the possibility of adding yet another layer to the production hierarchy without having to alter the entire database.

The *Products* table, together with the different product type tables, stores information about different product configurations. The *Products* table contains information shared between all types of products, such as ID and name, while the different type tables contain further type-specific information.

As mentioned earlier, a row within the *FactoryProducts* table serves as a bond between a factory and a product, creating what is called a "FactoryProduct". The reason for using such a bond is the fact that one product may be produced in one or more factories and reports for that product should be filtered based on which factory it was produced in. Thus rows within the *Deliveries* table refer to the *FactoryProducts* table rather than the *Products* table, since referring directly to the *Products* table instead would result in the merger of report data from different factories.

The two last tables, *Deliveries* and *OrderBook*, store reported delivery and order book data for a product produced in a certain factory, referring to the *FactoryProducts* table. The reason for using two separate tables is the amount of order book and reservation slot data compared to the amount of delivery data. Since the delivery data only counts for the current month while orders range monthly three years ahead, the amount of order book data becomes 36 times greater than the amount of delivery data. Therefore it was considered most logical to store every monthly order and slot reservation as one row separately in the *OrderBook* table, while referring to the "DeliveryID" of the respective row in the *Deliveries* table. In this way one can simply fetch the entire set of delivery, order book and slot reservation data based solely on the ID of one delivery row.

4.2.3 Security

As is always the case when it comes to confidential database information, security is a key criterion, but in this case the company servers were already under strong protection, thus the implementation of any advanced protection for the database alone was not necessary. It was still considered a good idea to implement some kind of encryption for the user passwords stored within the database.

When encrypting data, there are several encryption functions built into the SQL language that can be accessed using the HASHBYTES method. A popular one is the MD5 cryptographic hash function which produces a 128-bit hash value out of any given string (*Code example 6*). A password can safely be stored using MD5 as any password string entered inside the GUI is immediately hashed before stored into the database. It does not matter that MD5 cannot be decrypted, as the same given string will always generate the same hash value and thus the hash value of a password string entered in the GUI can be compared to the stored hash values in the database to verify the user's credentials. [15]

Code example 6. Two examples showing results of the MD5 function used in PHP. As can be seen, the length of the given string does not make any difference to the length of the resulting hash string.

```
MD5("The quick brown fox jumps over the lazy dog")
= 9e107d9d372bb6826bd81d3542a419d6

MD5("")
= d41d8cd98f00b204e9800998ecf8427e
```

4.3 Application

4.3.1 Intended functionality

There should essentially be three different types of actors accessing the OBVTT: the *Visitor*, the *User* and the *Administrator*. The *Visitor* should be an anonymous person accessing the application via the Wärtsilä intranet, while the *User* and the *Administrator* should be employees who have signed into the application and acquired certain permissions. A thorough explanation of each actor's role follows below, with a use case diagram visualizing the functionality in *Figure 8*.

The *Visitor's* permissions should be restricted to signing in to the application, requesting a new password and accessing some of the help contents. Once the visitor has signed in, he or she should become either a *User* or an *Administrator*. Every tool available to the user should be shared by the *Administrator*, thus the *Administrator* could be considered an extension of the regular user. The shared tools should consist of viewing and editing the personal account, accessing help and resources, and naturally the ability to sign out (although a signed-in user should automatically be signed out by the end of a session). Furthermore both the *User* and the *Administrator* should be able to report and view delivery and order book data, based on the credentials assigned to location and product for each user.

In addition to the shared tools, the *Administrator* should obviously have access to unique administrative tools, which allow him or her to manage every aspect of the application. These tools should allow the *Administrator* to manage user accounts, user credentials, product centers, factories and products, controlling what data each user may access. The *Administrator* should also be able to manage the resources available to regular users.

Finally there should be a *Database administrator* allowed to access the actual database using special database management tools. Such a user could set up regular backups, monitor the database if needed and even override some table values in critical situations.

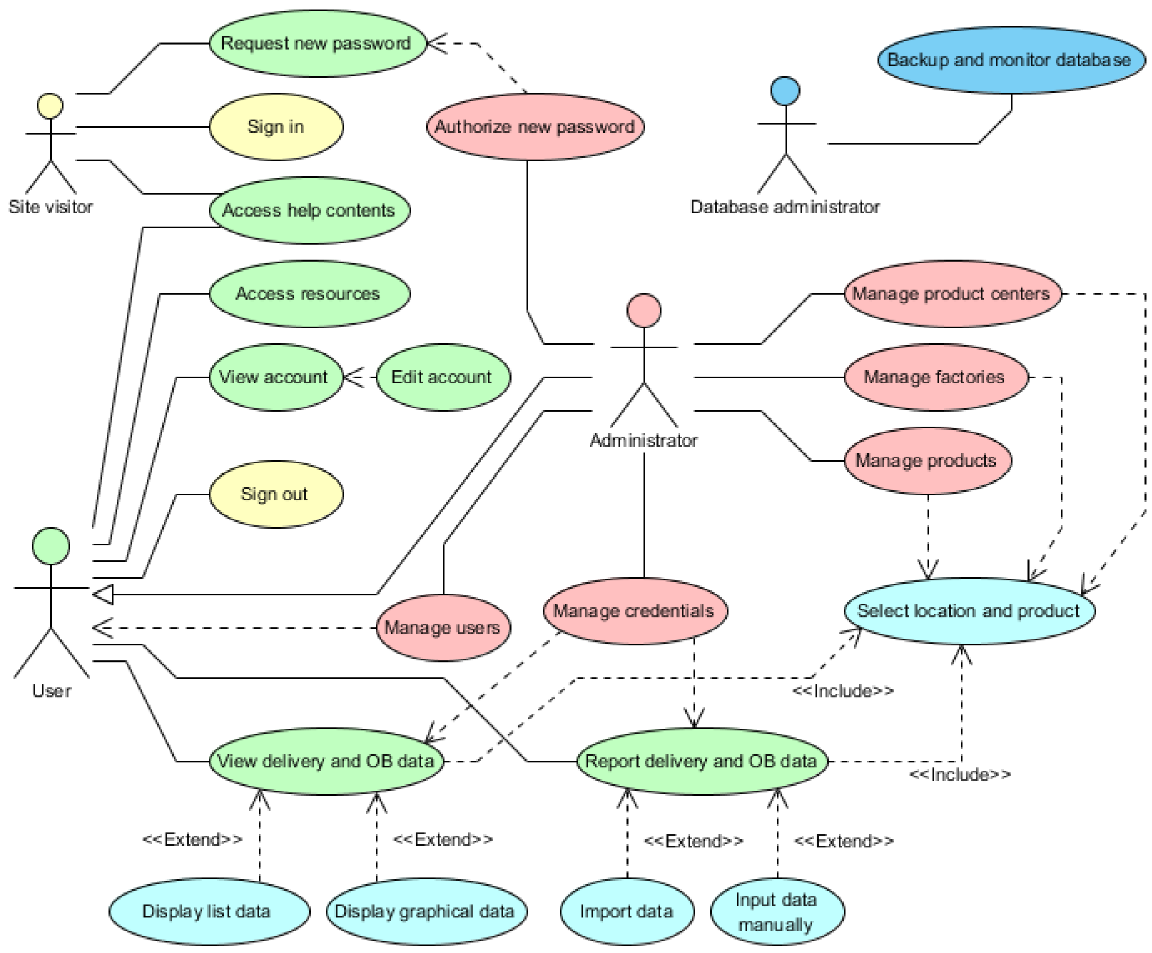


Fig 8. OBVTT use case diagram.

4.3.2 Connecting to the database

ASP.NET offers several ways to interact with different types of databases. A popular way to connect to a database and its tables is using *Entity Framework (EF),* an ADO.NET based object-relational mapping framework that eliminates impedance mismatch between data models that might occur with new versions of ADO.NET. Early versions of the project contained an Entity Framework Model connected to the database, but due to the structure of the application it was later decided to switch to manual ADO.NET coding. [16]

A separate class (*OBVTT_DB*) was created for interaction between the application and the database, containing all the required methods for reading and writing data, while the connection string to the database was handled in the constructor. A static instance of the *OBVTT_DB* class was made available in the *Global* file inside the project. In this way the object could be called from any page within the project without the need of defining a new *OBVTT_DB* object in every single file.

The *OBVTT_DB* class contains typical ADO.NET methods for selecting, inserting, updating and deleting data stored within the database. There are methods for returning all items of a certain type as well as methods using parameters such as ID to filter the results, returning only the desired items. The *OBVTT_DB* also contains methods for verification of data, server status checking and other database-related interactions.

The methods used in the *OBVTT_DB* class are very much alike, as most of them simply execute commands to the database and return some value or objects, while others are more advanced, using transactions or calling other methods within the method. An example of such a method can be seen in *Code example 7*. By including certain namespaces, different important objects become available, such as the *SqlCommand* and *SqlDataReader* seen in the example. The *SqlCommand* is assigned an SQL query string, which may also include different parameters for filtered database interaction if needed. Once a connection to the database has been opened, an *SqlDataReader* reads and gathers the requested data, which is then inserted into the correct type of object, in this case *CUser* (explained in subchapter 4.3.3). Finally the connection is closed and the method returns the entire list of objects.

Code example 7. A simple method using an SqlDataReader to read all data from the Users table in the OBVTT database.

```csharp
public List<CUser> GetUsers()
{
    SqlCommand comm = new SqlCommand();
    comm.Connection = conn;
    comm.CommandText = "SELECT * FROM Users ORDER BY lastName";

    conn.Open();

    SqlDataReader reader = comm.ExecuteReader();
    List<CUser> users = new List<CUser>();

    while (reader.Read())
    {
        users.Add(new CUser((int)reader[0], reader[1].ToString(),
            reader[2].ToString(), reader[3].ToString(),
            reader[4].ToString(), Convert.ToBoolean(reader[5])));
    }
    conn.Close();

    return users;
}
```

4.3.3 Classes and interfaces

An ASP.NET application usually contains many classes from scratch, serving for fundamental parts of a project, such as content pages and global variables. To make the project properly reflect the database data, additional custom classes were created to represent rows from each table in the database. Two main classes, *CProduct* and *CSector*, represent rows from the database tables *Products* and *Sectors*, while separate child classes inherited from these two represent related tables in the database. These classes contain some variables equivalent to most columns, as well as different properties and methods designed to streamline the application. Due to the many properties of an engine, a couple of enumerators were added to more easily keep track of certain attributes. A representation of the application class hierarchy can be seen in *Figure 9*.

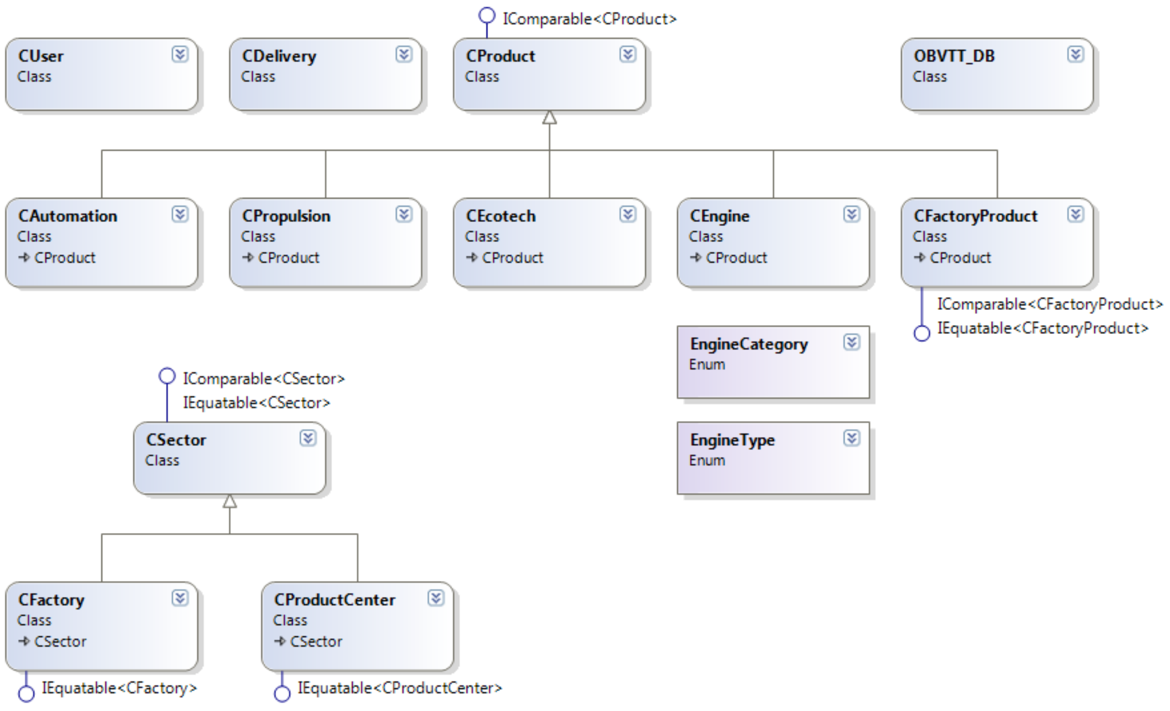


Fig 9. OBVTT class diagram.

A third main class *CUser* was created to represent a signed-in user. Similarly to other custom classes in the project, *CUser* reflects the row of a table within the database, in this case the table *Users*. The *CUser* object could be considered one of the most important parts of the project, as it represents the user's identity and credentials, completely controlling the user's access within the application. Whenever a user signs in to the application, the object is created based on his or her attributes, and will remain as a session variable throughout the session, unless the user signs out. Although the *CUser* object is the factor deciding the accessibility of the GUI, the user always undergoes validation (interfacing directly with the database via the *OBVTT_DB* class) whenever he or she is about to read or write data. This prevents any kind of credential breach, should the user somehow have accessed restricted areas of the application.

The final custom class created was *CDelivery*, which, unlike previous classes, represents two database tables: *Deliveries* and *OrderBook*. This is due to the fact that the two tables contain very similar data, dates and amounts, easily fitting into one class. In the application each delivery is represented by one *CDelivery* object which contains two lists of yet more *CDelivery* objects, the orders and reservations that can be accessed as properties. This allows for a complete package of report data contained in one object.

Two different interfaces (*IComparable and IEquatable*) were applied to some of the classes in order to enable the use of certain methods available in the .NET framework. These interfaces render an object comparable to another, using different methods of comparison. Using interfaces greatly reduces the amount of code needed as one avoids having to create methods for every small task required in the code, such as removing duplicates from an array or a list object, which can be done using the *Distinct* method which in turn uses *IEquatable* for comparison.

4.3.4 Binding data

As the application contains many lists and dropdown boxes, it must naturally somehow connect the database data to these components. There are many ways to manually achieve data binding, such as adding a list of objects to the component's collection array, but the more practical alternative is to use some type of data source component. For the OBVTT, object data sources were used to bind data to components.

The object data sources used in the OBVTT are assigned different selection methods. These are the user created methods available within the *OBVTT_DB* class, the database class mentioned earlier in this chapter. When the methods are called, they acquire data from the database and create lists of objects which are then returned to the data source

(*Code example 7*) and finally to the component, assigning a value and data text field to each item in the component's collection. Whenever the user then selects an item, the selected value can be read and used as a parameter for further data binding or other uses.

4.4 User interface

4.4.1 Main design

The design is heavily based on previous iterations of Wärtsilä web applications. It consists mostly of CSS rendered graphics along with different ASP.NET and HTML components. A few images were also created, mostly to add a little style to the design. The color scheme is entirely based on the official Wärtsilä color palette, which leans very much towards gray and orange.

The layout consists of two main parts: an orange header running across the top of the page, containing the application title and the main menu, as well as the content area below, in which content is presented within gray panels with rounded corners. Elements such as panels are all dynamically sized to allow a smooth presentation of the application whether it is displayed on a huge desktop monitor or on a tiny laptop display. To allow for dynamic resizing one can simply assign percentual size values to an element using CSS. Unfortunately some of the design is limited, based on browser versions, as older browsers are not compatible with newer CSS capabilities.

The CSS classes used throughout the application are stored in a separate CSS file called "style.css". The file contains many different custom classes, as well as alterations of HTML elements. Examples of CSS classes used in the project can be seen in *Code example 8*.

Code example 8. Two CSS classes, the former being a custom class that can be assigned to any element, while the latter alters the h1 HTML element.

```
.content_style{
    padding: 20px 200px 0px 30px;
}

h1{
    font-size: 1.6em;
    padding-bottom: 0px;
}
```

4.4.2 Navigation

As is common in web pages and web applications, a main menu bar is usually always available somewhere on the page to allow for quick access to the most important sections of the application. In the OBVTT the menu bar is an *<asp:Menu>* element that runs horizontally across the bottom of the page header. Whenever the user clicks on one of the menu items, he or she then navigates to a new URL.

Each page available from the main menu usually contains one or more menus of its own. While some of these menus, much like the main menu, transfer the user to a new URL, others simply enable and/or disable different elements and components on the page through the use of events.

4.4.3 Presentation of information

When the user navigates to a new page within the application, he or she is always presented with one or more panels containing some information or components. If the user interacts with these components, more panels will appear, requesting new input from the user. This means that only the information that is relevant at the moment is visible, helping the user navigate through each tool without having to constantly search for the input to use.

Apart from some common HTML elements such as line breaks, headers and tables, the code consists mostly of ASP.NET components. A list of the most important and frequently used components (excluding subcomponents) can be seen below.

- *<asp:Button>*
- *<asp:Content>*
- *<asp:DropDownList>*
- *<asp:ListBox>*
- *<asp:Menu>*
- *<asp:ObjectDataSource>*
- *<asp:Panel>*
- *<asp:TextBox>*

To notify the user of successful actions, errors or other important information, a notification bar floating to the left side of the page was added using absolute positioning in CSS. This bar appears and disappears automatically over time, but may also be hidden immediately by being clicked on whenever visible.

4.5 Testing

Being a web-based application, SuperPreview, along with a range of browsers, was used for testing throughout development. The main focus was on Internet Explorer and Google Chrome, as these browsers were most frequently used within Wärtsilä. However, it was obviously crucial to have a fully functional and correctly rendered application running on any type of browser.

The most common deviations noticed between the different browsers were different layout issues. While some were minor, only causing some cosmetic anomalies, others would result in non-functioning components or text being offset to the point where it became undecipherable. All issues regarding functionality were naturally addressed and fixed, but some of the minor cosmetic issues remain as they only somewhat change the appearance rather than ruin it.

5 The product

5.1 Main view

Whenever a user accesses the web application, he or she is presented with the *Main view*. The *Main view* welcomes the user and may present news or updates regarding the application or the server, as well as database status. Just like every other view within the application, the *Main view* consists of a menu bar at the top of the page, from where the user can access other views as well as the different tools. Depending on credentials and whether the user is signed in, different views might be unavailable at the time being.

5.2 Account view

The *Account view* is first in line of the items accessible from the main menu bar, serving as both an interface for signing in, as well as a tool for managing the account. Depending on whether the user is signed in or not, the title of the menu item may change to better display which user is currently signed in. *Figure 10* displays a user signing in.

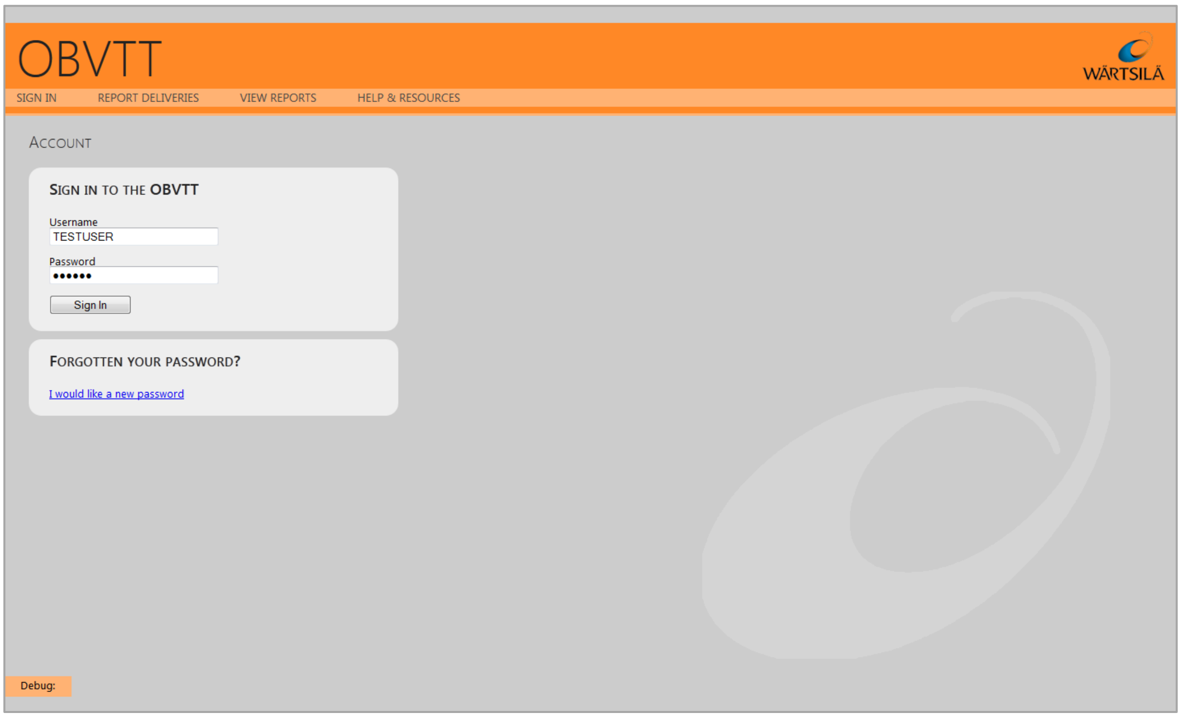


Fig 10. The Account view of the OBVTT requesting the user to enter his username and password.

5.2.1 Signing in

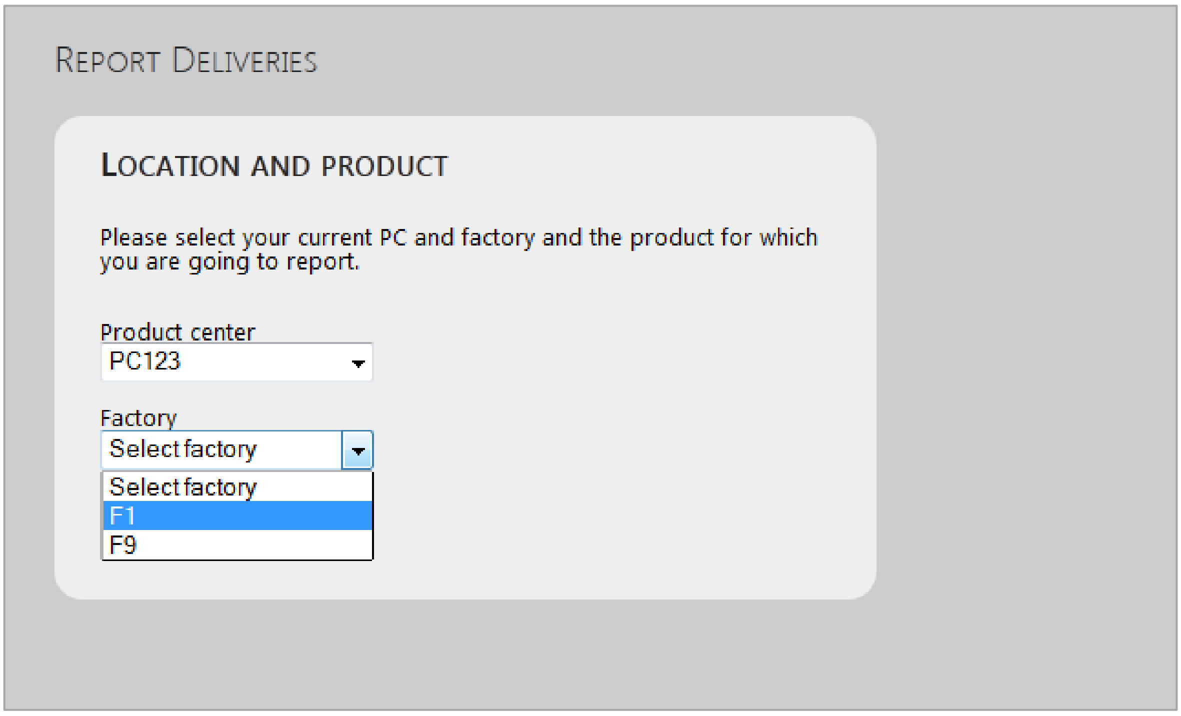
In order to access the actual tools, the user must sign in to the application via the *Account view*. The *Account view* does not initially offer much functionality past the input boxes used for signing in, apart from the possibility to restore a forgotten password, should the user need to. Should the user decide to request a new password, an e-mail client is opened with a password request addressed to administrators for the user to submit.

5.2.2 Account management

Once a user has signed in to the application, the *Account view* displays account management instead of tools for signing in. Within the *Account view* the user is able to view and edit his or her account. For instance the user may change personal information or the password associated with the account. Changing passwords obviously requires input of the current password, which as the standard is when managing personal credentials.

5.3 Reporting delivery and order book data

The *Report deliveries view* gives the user access to one of the two main tools available in the OBVTT. When a user is ready to submit delivery and order book data for one or more products, he or she may open this view, choose the appropriate product center, factory and product using dropdown-boxes (*Figure 11*), and select an input method. The user may either manually input the data into a table or import an Excel file containing the delivery data (*Figure 12*). Depending on the user's credentials, he or she may only have access to a few of the product centers, factories and/or products available in the OBVTT database, limiting his or her options accordingly. Administrators on the other hand may report data for any product in the database.


REPORT DELIVERIES

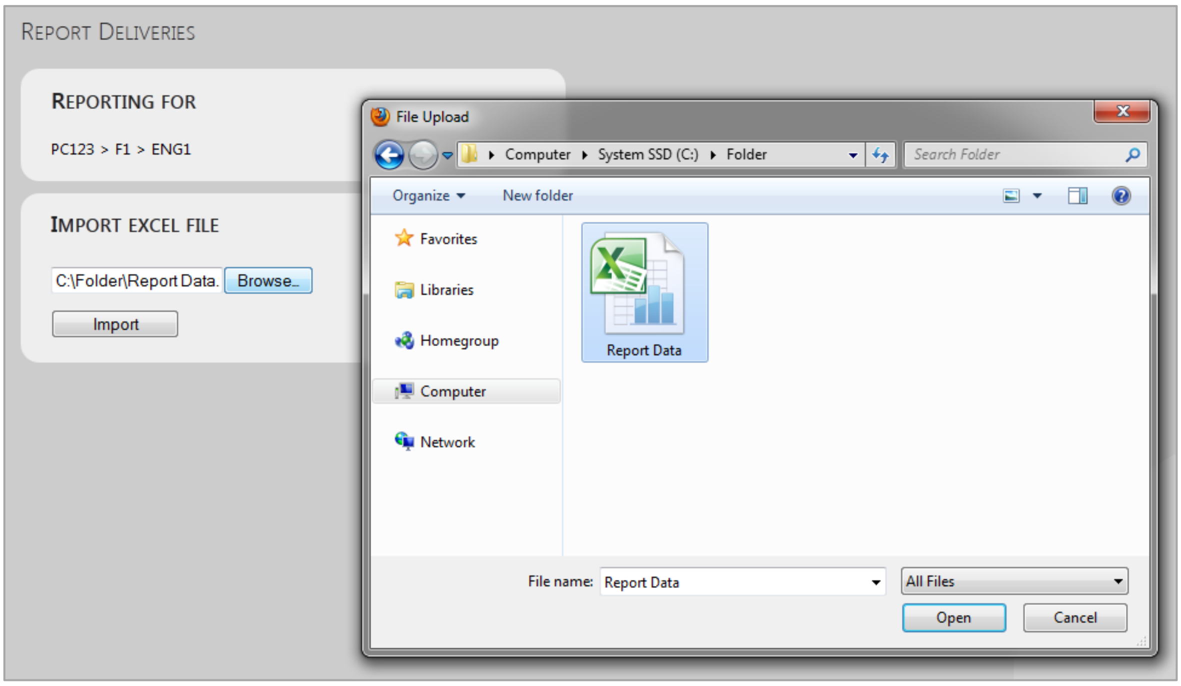
LOCATION AND PRODUCT

Please select your current PC and factory and the product for which you are going to report.

Product center: PC123

Factory: Select factory

Fig 11. The user selects the product for which he or she would like to report data, based on location.


REPORT DELIVERIES

REPORTING FOR

PC123 > F1 > ENG1

IMPORT EXCEL FILE

C:\Folder\Report Data.

File name: Report Data

All Files

Fig 12. Importing an Excel file containing report data. The application verifies the type and content of the file before accepting it.

When submitting data by importing an Excel file, the user is required to use a certain template file. The template follows Wärtsilä standards and is available for download via the *Help & Resources view*.

Once the user has submitted his or her report, the data is immediately stored inside the OBVTT database and may be reviewed using the *View reports tool*. Should the user realize that an error was made while reporting, he or she may edit or completely remove the report altogether, given that the user is responsible for that data. The data is open for editing until the user or an administrator decides to lock it.

5.4 Viewing delivery and order book data

The *View reports view* gives the user access to the other main tool available in the OBVTT. In order to track and analyze deliveries and order book date for a certain product, the user may filter data accordingly and receive a presentation based on chosen data. As mentioned about the *Report deliveries tool*, the user may only have access to certain data depending on credentials. Administrators, however, have full access to all stored data.

The delivery and order book data chosen for presentation may consist of both actual data tracked over a desired timespan and future prognoses. Also depending on whether the attributes of the product have undergone changes during its lifetime, the user may filter and access historical data for different configurations. Once the user has rendered the desired chart image, he or she may either save it as an image file or print it.

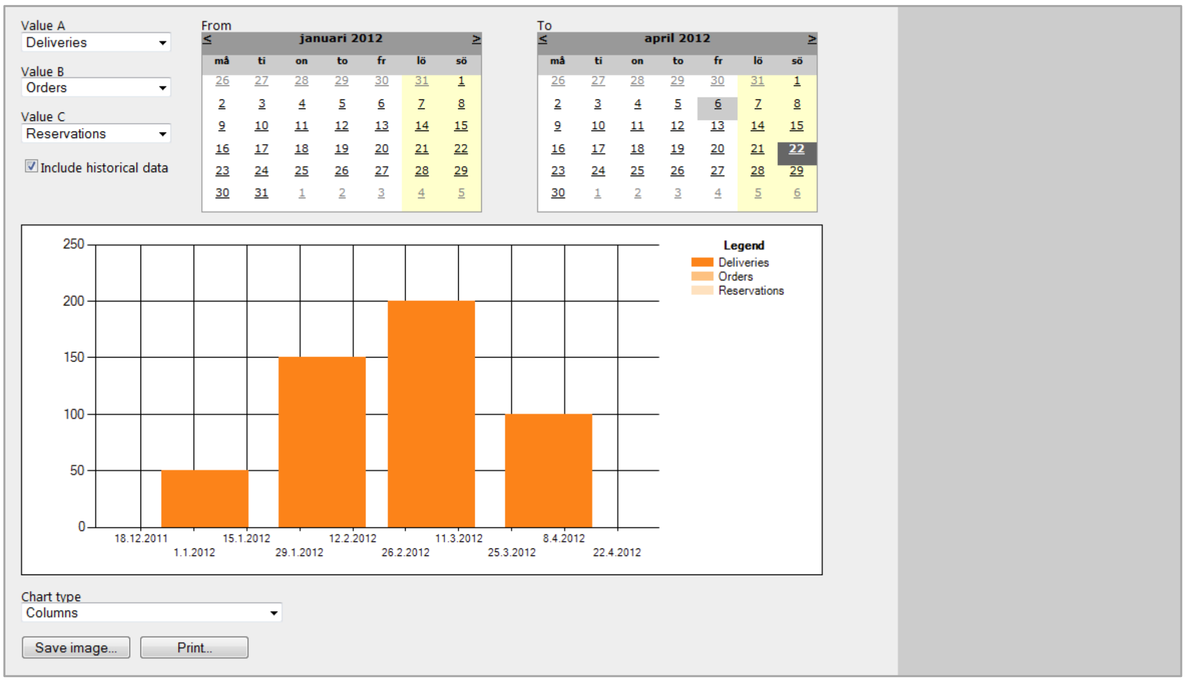

Value A: Deliveries
Value B: Orders
Value C: Reservations
☑ Include historical data
From: januari 2012
To: april 2012
Chart type: Columns

Fig 13. Viewing delivery and order book data for a certain product. The user has access to several tools allowing him or her to filter data and customize the chart.

5.5 Help and resources

To ensure that anyone can use the different tools without any type of training, detailed instructions on how to use every part of the application are available through the *Help & Resources view* which, like any other section, can be found in the main menu bar.

In addition to different instructions and help manuals, the *Help & Resources view* offers different resources that might prove useful when using the OBVTT. A good example of such a resource is the Excel template used for reporting delivery and order book data.

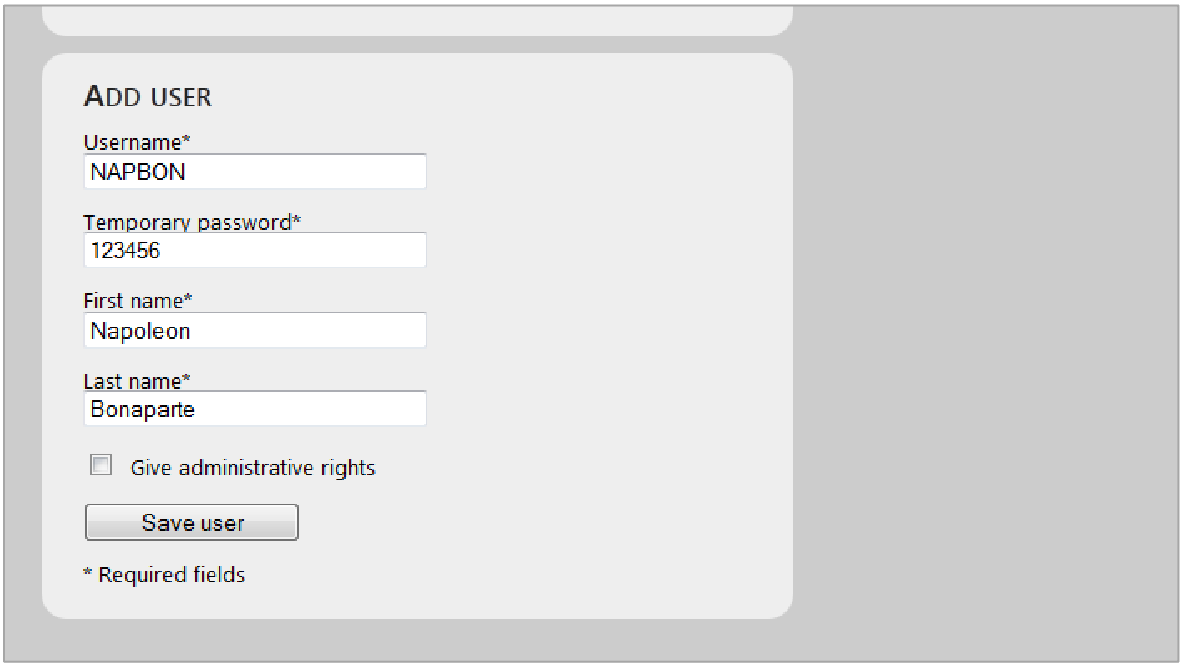
Finally the *Help & Resources view* displays some information about the application itself, the developer and some help contact details, should the help contents not suffice.

5.6 Administrative tools

Should a signed in user have administrative rights, this user may access the administrative tools of the OBVTT. The administrative tools allow the administrator to edit base data in the database, such as user accounts, product attributes, product centers and factories.

5.6.1 Managing users and credentials

An administrator has the rights to add new user accounts, modify existing ones and deactivate obsolete ones (*Figure 14*). Once a user has been created, the administrator may assign different credentials to that user by opening the *Manage user credentials tool* and simply selecting which product centers, factories or products the user should have access to and the type of permissions (*Figure 15*). The user being managed may gain read or write permissions for products, factories or entire product centers. Gaining permissions to edit an entire product center would for instance mean that the user may edit any product being manufactured in the factories belonging to that product center. Naturally these permissions can be changed at any time.


ADD USER

Username*
NAPBON

Temporary password*
123456

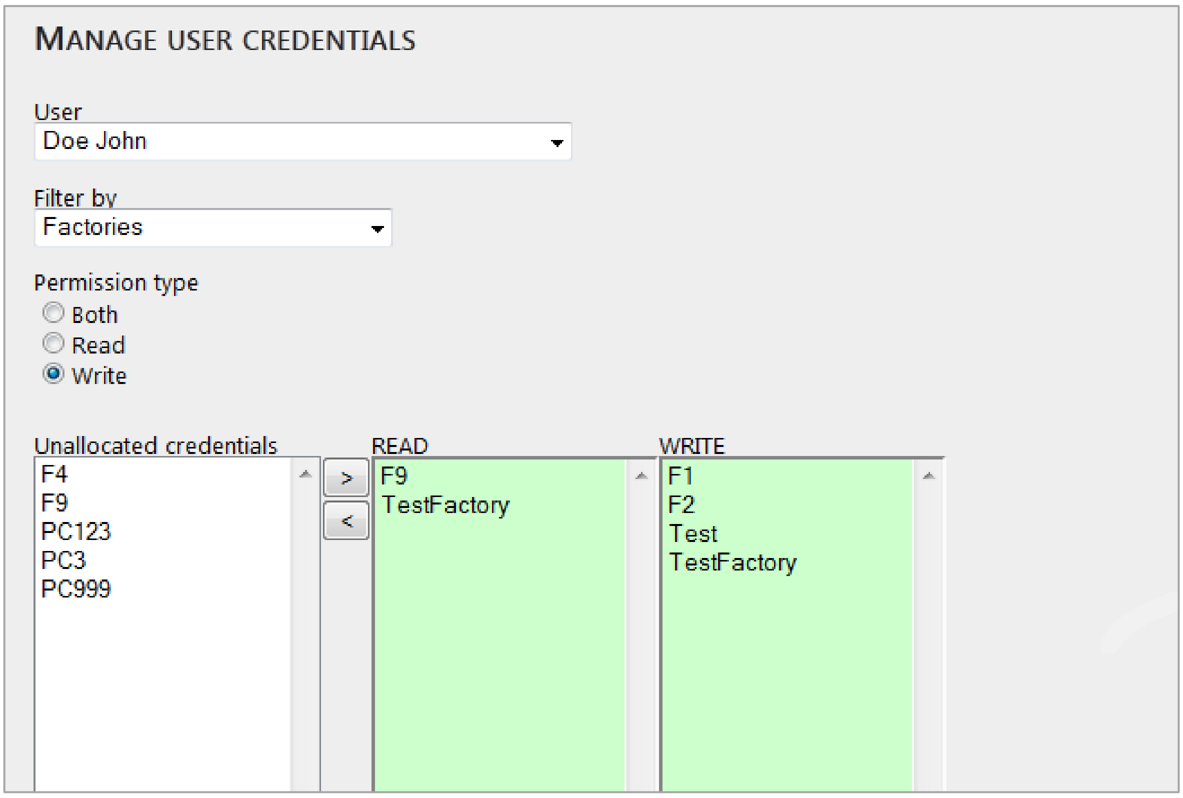
First name*
Napoleon

Last name*
Bonaparte

☐ Give administrative rights

* Required fields

Fig 14. Adding a new user.


MANAGE USER CREDENTIALS

User
Doe John

Filter by
Factories

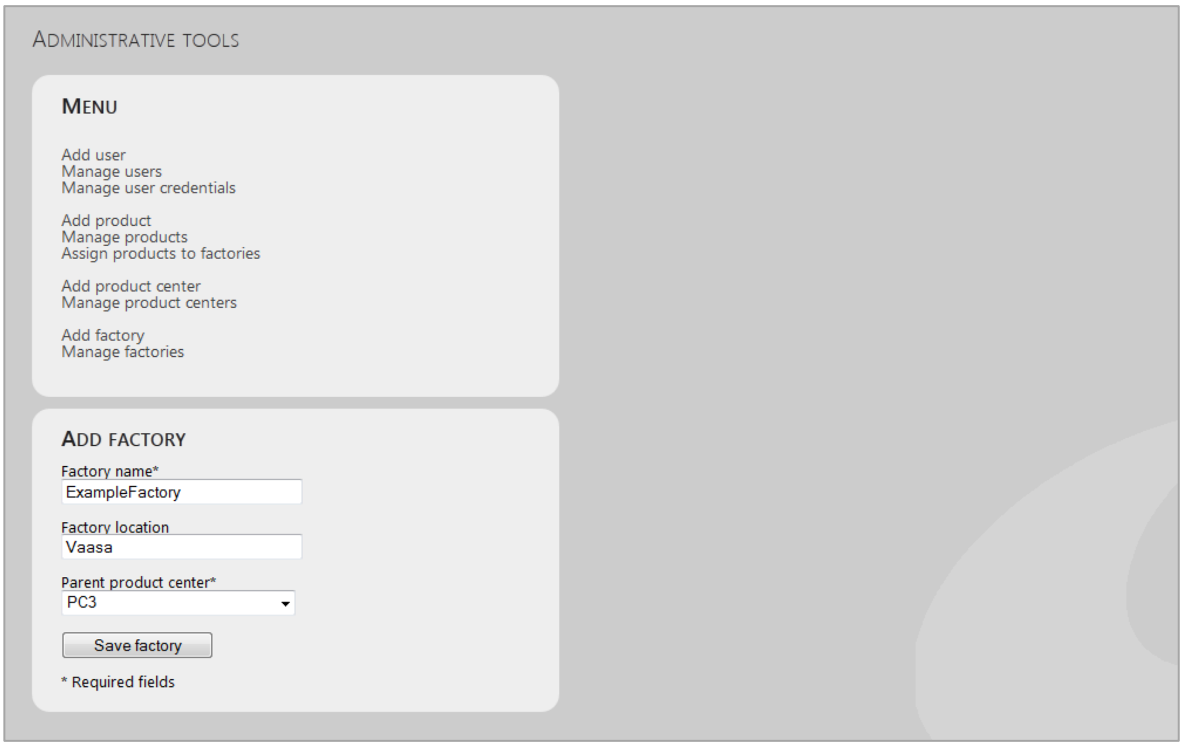
Permission type
○ Both
○ Read
● Write

Unallocated credentials	READ	WRITE
F4	F9	F1
F9	TestFactory	F2
PC123		Test
PC3		TestFactory
PC999		

Fig 15. Managing user credentials.

5.6.2 Managing product centers, factories and products

Just like an administrator may add and modify user accounts, he or she may also add or modify product centers, factories and products (*Figure 16*). However, whenever the attributes of a product are altered, a new product containing the assigned attributes is actually added to the database. Meanwhile the modified product is deactivated but left to remain in the database. The reason why the old product attributes are kept is to maintain historical data for different products, as the new attributes do not apply to past deliveries and orders. This allows filtering of historical product specifications when viewing delivery and order book data.



ADD FACTORY

Factory name*
ExampleFactory

Factory location
Vaasa

Parent product center*
PC3

* Required fields

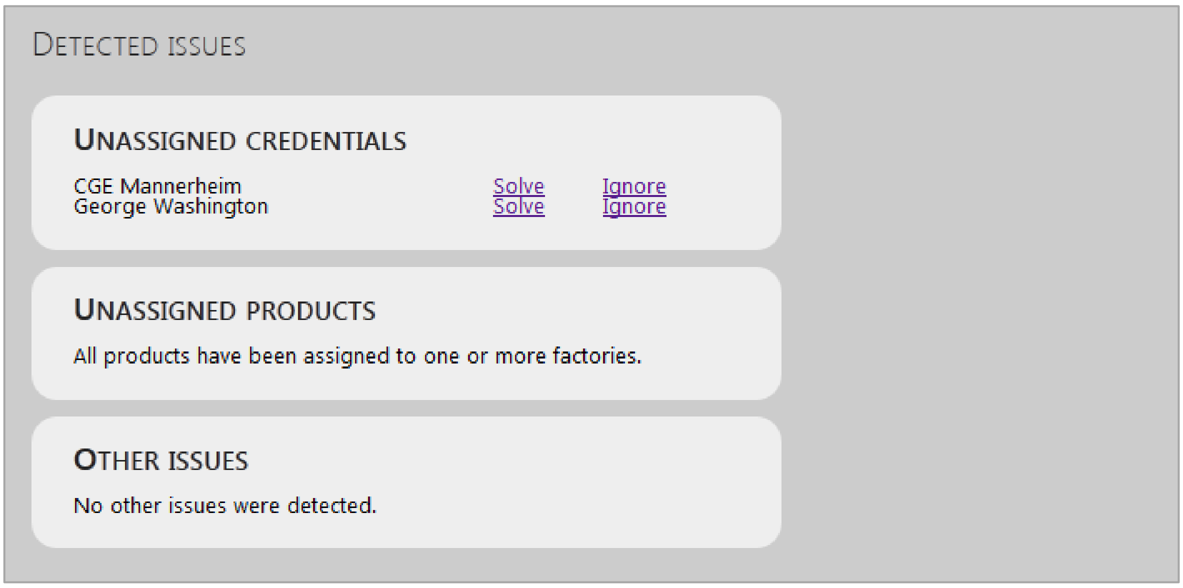
Fig 16. Adding a factory using the administrative tools of the OBVTT. Since each factory belongs to a product center, the user must choose which product center the new factory is to belong to.

5.6.3 Application-specific settings

The final tools available to administrators are application specific settings that affect the GUI rather than the database. These tools allow an administrator to dynamically modify some of the GUI appearance and manage downloadable resources and information. It also lets the administrator send messages to the application’s main view, in case users of the OBVTT should be notified of any changes or other important events.

5.6.4 Warning system

Since credentials and bonds between factory and product must be manually assigned, the risk of leaving some users, products or factories unassigned is fairly high. To avoid this there are several verification methods run frequently to ensure that everyone and everything is connected. If an object fails verification, the administrator is alerted by an "ALERT!" item appearing in the main menu bar. This item directs the administrator to a page where every located issue is listed (*Figure 17*), with information on how to solve each problem. This removes the need to manually verify each object using the different management tools and works as both an alarm and a time saver.


DETECTED ISSUES

UNASSIGNED CREDENTIALS

CGE Mannerheim Solve Ignore
George Washington Solve Ignore

UNASSIGNED PRODUCTS

All products have been assigned to one or more factories.

OTHER ISSUES

No other issues were detected.

Fig 17. The warning system displaying detected issues.

6 Discussion and feedback

6.1 Thoughts on the product

The product is a functioning tool that allows employees to report and view delivery and order book data, as was the main requirement. The application has not yet been taken into use by Wärtsilä Industrial Operations as it is still undergoing testing and optimization and will also undergo some heavy design updates before being taken into use. It is targeted for implementation within the year.

Like any application, the OBVTT is naturally prone to the risk of bugs or other errors showing up unexpectedly in the future despite thorough testing and debugging. It is naturally possible to backup any important data and quickly locate and repair the problem. The application should be supported by the developer as long as it remains in use.

6.2 Future development

Although the product is considered fully functional, there is naturally always room for improvements and the addition of new features that enhance the application either visually or practically.

6.2.1 Connecting the user accounts to Active Directory

The application is currently using its own account system, with user accounts being stored in a database table. It would, however, be a great improvement to connect the user accounts of the application to the Active Directory accounts used throughout the Wärtsilä internal network. In this way, users would not have to keep track of many different accounts and would be able to sign in automatically when accessing the application.

6.2.2 Enhancing the application with Silverlight

Currently the application consists exclusively of HTML and ASP.NET components. Although this provides necessary functionality, implementing some Silverlight plug-ins would allow for a much more graphically capable user interface, especially when viewing report data.

Microsoft Silverlight is a free web-browser plug-in similar to Adobe Flash, that enables different interactive media in the web browser, such as applications, games or videos. Silverlight works on all major operating systems and all major web browsers, proving to be a very versatile plug-in. [17]

6.3 Reflections

The OBVTT is probably the most advanced project I have been assigned thus far as a developer. Not having designed any software for such a large corporation before, I was rather nervous as to whether I could really handle the task. At first it seemed very difficult, trying to get a grasp of the ecosystem of Wärtsilä, the role of employees, the production phases etc. I soon realized that taking detailed notes would be crucial and it definitely helped me over time to understand how I would have to design this application to provide desired functionality. The people I got to work together with at Wärtsilä Industrial Operations were very eager to provide me with the resources I needed during the development process and I consider the time spent developing the OBVTT a great experience.

Developing the OBVTT has definitely been very rewarding to me as a developer, as I got the chance to use all the things I had learned over the years at Novia to create something advanced and important. As I had not previously developed any ASP.NET applications, it was also a good opportunity to become familiar with the platform and I soon grew very fond of its tools and features. Despite having previously developed several web sites and web applications using PHP, I would definitely prefer using ASP.NET from now on.

7 Table of sources

[1] "Wärtsilä," [Online]. Available: http://www.wartsila.com. [Accessed 27 February 2012].

[2] "Wärtsilä on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/W%C3%A4rtsil%C3%A4. [Accessed 6 April 2012].

[3] "Wärtsilä Industrial Operations Internal Presentation," 2012.

[4] "Microsoft SQL Server on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_SQL_Server. [Accessed 20 March 2012].

[5] "Free Database Software," [Online]. Available: http://www.microsoft.com/sqlserver/en/us/editions/express.aspx. [Accessed 6 March 2012].

[6] "SQL on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/SQL. [Accessed 20 March 2012].

[7] "Transact-SQL on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Transact-SQL. [Accessed 29 March 2012].

[8] "Visual Studio Home," [Online]. Available: http://www.microsoft.com/visualstudio/en-us. [Accessed 28 March 2012].

[9] "ASP.NET on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/ASP.NET. [Accessed 20 March 2012].

[10] "C# Language Specification," [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf. [Accessed 16 March 2012].

[11] "HTML on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/HTML. [Accessed 20 March 2012].

[12] "Web-forms: The Official Microsoft ASP.NET Site," [Online]. Available: http://www.asp.net/web-forms. [Accessed 26 March 2012].

[13] "CSS on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Cascading_Style_Sheets. [Accessed 20 Match 2012].

[14] "Paint Shop Pro on Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Paint_Shop_Pro. [Accessed 29 March 2012].

[15] "What is the MD5 checksum?," [Online]. Available: http://www.fastsum.com/support/md5-checksum-utility-faq/md5-checksum.php. [Accessed 29 March 2012].

[16] "The ADO.NET Entity Framework Overview," [Online]. Available: http://msdn.microsoft.com/en-us/library/aa697427%28v=vs.80%29.aspx. [Accessed 29 March 2012].

[17] "About | Microsoft Silverlight," [Online]. Available: http://www.microsoft.com/silverlight/what-is-silverlight/. [Accessed 29 March 2012].

[18] M. MacDonald, A. Freeman and M. Szpuszta, Pro ASP.NET 4 in C# 2010, 2010.

M. MacDonald, A. Freeman and M. Szpuszta, Pro ASP.NET 4 in C# 2010, 2010.

R. Dewson, Beginning SQL Server 2008 for Developers, 2008.