A WCF service implementation of a statistics logger prototype

En WCF service implementation av en prototyp av statistik loggare

Svensson, Kenny
Gustafsson, Andreas

Faculty: Faculty of Health, Science and Technology
Subject: Computer Science
Points: 15hp
Supervisor: Leonardo Martucci
Examiner: Donald F. Ross
Date: 2015-06-01
A WCF service implementation of a statistics logger prototype

Kenny Svensson
Andreas Gustafsson

© 2015 Kenny Svensson, Andreas Gustafsson and Karlstad University
This thesis is submitted in partial fulfilment of the requirements for the Bachelor’s degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

__________________________
Kenny Svensson

__________________________
Andreas Gustafsson

Approved, 01/06/2015

__________________________
Advisor: Leonardo Martucci

__________________________
Examiner: Donald F. Ross
Abstract

This thesis project is about collecting usage statistics in an application, how the information can be logged and presented in a suitable way and to what purpose these statistics are used. The project is a logging service to which applications can connect and send performance and usage data. The service logs the data to a file, which can be opened in QlikView for presentation of data statistics.

The resulting statistics are planned to be used for the evaluation and improvement of the client application and improvement on the end users way of working with that application. The service was implemented as a WCF service in C# and uses the logging capabilities of the Enterprise Library Logging Application Block. A WCF service is a service capable of having a variety of different applications connected to it, using different communication protocols.

The result of our project is a working service prototype which logs data to file. Data generated by a test client is successfully logged by our service and its statistics is displayed in an application named QlikView.
Table of Contents

Chapter 1 - Introduction ............................................................................................................. 1
  1.1 - Project goal ....................................................................................................................... 1
  1.2 - Motivation - Project impact on different systems ........................................................... 2
  1.3 - Contributions .................................................................................................................. 3
  1.4 - Result summary ............................................................................................................... 5
  1.5 - Chapter summary ............................................................................................................ 6
  1.6 - Organisation of the thesis .............................................................................................. 6
Chapter 2 - Background ............................................................................................................. 9
  2.1 - WCF Service ..................................................................................................................... 9
  2.2 - Enterprise Library Logging Application Block ............................................................... 11
  2.3 - The Test Client ............................................................................................................... 14
  2.3.1 - Silverlight .................................................................................................................. 16
  2.3.2 - XAML ........................................................................................................................ 17
  2.4 - QlikView ......................................................................................................................... 18
  2.5 - Development tools ......................................................................................................... 18
  2.6 - Chapter summary ............................................................................................................ 18
Chapter 3 - System prototype design and implementation ..................................................... 19
  3.1 - Requirements .................................................................................................................. 19
  3.2 - Design ............................................................................................................................. 21
  3.2.1 - WCF web implementation, WEB ............................................................................. 22
  3.2.2 - Business Logic Layer, BLL ..................................................................................... 22
  3.2.3 - Data Access Layer, DAL ......................................................................................... 22
  3.2.4 - Interface layer, Contracts ....................................................................................... 22
  3.3 - Implementation ............................................................................................................... 23
  3.3.1 - WCF web implementation, WEB ............................................................................. 24
  3.3.2 - Business Logic Layer, BLL ..................................................................................... 25
  3.3.3 - Data Access Layer, DAL ......................................................................................... 26
  3.3.4 - Interface layer, Contracts ....................................................................................... 27
  3.3.5 - The test client ............................................................................................................. 28
  3.4 - Chapter Summary .......................................................................................................... 28
Chapter 4 - Results ................................................................................................................... 29
4.1 - The Resulting prototype ................................................................. 29
4.2 - Project goal vs. result................................................................. 31
Chapter 5 - Conclusion............................................................................ 33
  5.1 - Summary...................................................................................... 33
  5.2 - Project Management..................................................................... 33
  5.3 - Future work / Next step .............................................................. 34
References ............................................................................................. 38
Appendix A - Code ............................................................................... 40
Appendix B - Solution chart................................................................. 50
List of Figures

Figure 1.1: A map over the data path from gathered till presented.......................... 2
Figure 1.2: Example of what can be written to the log on user login and logout ............ 4
Figure 1.3: Example of what can be written to the log on performing a search ............ 5
Figure 2.1: Example of a Client - Web Service communication [9]............................ 10
Figure 2.2: Example of a Client – Remoting Service communication [9].................... 10
Figure 2.3: A WCF service version for the client applications in Figures 2.1 and 2.2 [9] ... 11
Figure 2.4: An overview of the logging process in the Logging Application Block [3] .... 12
Figure 2.5: A view of our logging configuration.......................................................... 13
Figure 2.6: Code snippet of our logging method.......................................................... 14
Figure 2.7: The test client application that was used for testing the WCF-service............ 15
Figure 2.8: The test client application after execution of a simulated login and a search ... 16
Figure 2.9: Some of the XAML generated as the GUI is graphically constructed above ... 17
Figure 3.1: A map over the structure of our service.................................................... 21
Figure 3.2: Code snippet of the WCF implementation, StatisticsService.svc.cs........... 24
Figure 3.3: XML code for client access policy, clientaccesspolicy.xml.......................... 25
Figure 3.4: XML code for cross domain access, crossdomain.xml............................... 25
Figure 3.5: Code snippet of methods in the BLL, StatisticsLogic.cs........................... 26
Figure 3.6: Code snippet of the class in our DAL layer, StatisticsAccess.cs.................. 27
Figure 3.7: Code snippet of the interface, IStatisticsService.cs................................. 27
Figure 3.8: Code snippet of one of the data contracts, UserBehaviourInformation.cs.... 28
Figure 4.1: Example view of data presentation in QlikView...................................... 29
Figure 4.2: Example view of data presentation in QlikView...................................... 30
Figure 5.1: View of what the search page in EquipMan looks like................................ 34
Figure A.1: StatisticsService.svc.cs (WEB)............................................................. 40
Figure A.2: StatisticsLogic.cs (BLL).......................................................................... 41
Figure A.3: StatisticsAccess.cs (DAL)......................................................................... 42
Figure A.4: IStatisticsService.cs (Contracts)............................................................ 43
Figure A.5: UserStatisticInformation.cs (Contracts).................................................. 43
Figure A.6: UserBehaviourInformation.cs (Contracts).............................................. 44
Figure A.7: PerformanceInformation.cs (Contracts)................................................... 45
Figure A.8: StatisticsError.cs (Contracts).................................................................. 46
Figure A.9: The login functionality we implemented in the test client....................... 47
Figure A.10: The collecting of data that we implemented in the test clients search method... 48
Figure A.11: The asynchronous call-backs we implemented in the test client............. 49
Figure B.1: A chart over our Visual Studio solution for our service............................ 50
Chapter 1 - Introduction

This thesis is a part of the Bachelor’s degree in computer science which was conducted during spring 2015 with Prevas AB and Karlstad University.

Prevas AB is an IT-company that was founded in 1985: The Company’s main focus lies on Industrial-IT and embedded systems. Prevas has customers all around the world and their offices are located in four different countries.

The project is a service prototype that logs information about behaviour, user statistics and performance of an application. This service is able to work with a variety of applications, logging data about the user experience of the application that is applied to the service. The user experience data consists of performance values on application functionalities, how often specific application functionalities are used and to what extent the available user input options are used for these functionalities.

This chapter presents the project goal and motivation followed by our contribution and a summary of the results of the project.

1.1 - Project goal

The project goal is to implement a WCF (Windows Communication Foundation) service [1] prototype that works together with a test client, provided by Prevas, and logs functionality performance of the functionalities used by which user as well as usage statistics in the test client.

The data gathered from the application are saved in an XML-file or database, and the statistical results are presented in QlikView [2]. See Figure 1.1 for an image of the process. The data collected should tell us which user that triggered which operation and the performance of that specific operation. Our WCF service will be available to many different applications in the future.
1.2 - Motivation - Project impact on different systems

The reason for collecting this information is to better understand how a user works with the application, what users need, what customers want, and to help to redesign user interfaces to make the system more intuitive and user friendly.

This information can be used for system evaluation and follow up on the way of working with the application, i.e., to advise the user on how to do certain tasks and to help with quality evaluation. The data can perhaps also be used for informal competitions in certain areas in the form of “employee of the month”, i.e., the user with the quickest searches on average per criteria used.
1.3 - Contributions
Our service was achieved by designing and implementing a WCF service in C# that receives statistical data. The service is made up of several layers, each with a specific task to perform:

- WCF web implementation - Receives data gathered from the client and sends it forth to our Business Logic Layer.
- Business Logic Layer - Builds an XML string from the data and sends that string to the Data Access Layer.
- Data Access Layer - Initiates a logger and sends it the XML string as input.
- Interface layer - Provides the interface for the WCF web implementation and has the classes all the layers needs to reference.

We make use of Enterprise Library Logging Application Block [3] in the Data Access Layer to easily write a XML string to a log file. The log file can then be opened in QlikView and the data can then be presented in different ways, e.g., as a chart or table.

We also implemented methods and a couple of GUI elements in the test client to generate data for our service to receive and thus be able to test our service. Different collections of the data would be set up in the client application and then sent to the service for processing.

Our service logs performance information, e.g., how long it takes for a user to get logged in or out. Our service logs statistical information on how long the user has been logged in. See Figure 1.2 for an example and take note of the tags “FunctionName” and “LogType”.
When performing an operation, for instance a search operation, our service logs user behavioural info for that operation. See Figure 1.3 for an example.
Before designing and implementing our service we performed a literature survey on WCF services and inspected the code of a WCF service Prevas had implemented. In collaboration with our supervisors at Prevas, we also analysed what kind of data that would be of interest for the company and that would be useful for future development of the applications. See section 3.1 for a list of data. During the implementation of our service we continuously tested it with a test client.

1.4 - Result summary

We have implemented a working WCF service that can receive and log information on how a user is using the client that is connected to the service as well as information on the performance of the client.

We generated data in the test client for our service to log and its statistics was presented in an application called QlikView. See section 2.4 for QlikView and see our results in section 4.
The most important result for us is that we got the test client to connect to the WCF service and to understand why it is such a good tool for us to use. We believe that we absolutely will use WCF services in our future work, especially now that we know the advantages of such a service.

1.5 - Chapter summary
In this chapter we gave a brief presentation of the project, its goal and motivation. We have implemented a WCF service that is able to log information, of how our test application is used, into an XML file. In the next chapter you can read more about why we use a WCF service and how it works.

1.6 - Organisation of the thesis
The rest of this thesis is organized as follows:

**Chapter 2** - In this chapter, we introduce WCF Services and some of its benefits, Enterprise Library Logging Application Block, what it does and how it works. We explain the test client and how it was used. We also introduce Silverlight [4], and XAML [5]. Lastly, we make a short introduction to QlikView, an application for data presentation, before presenting the development tools we have used.

**Chapter 3** - In this chapter we explain how we have modelled and implemented our service, starting with its requirements and the thought process behind the information classes and their properties. We then move on to an overview of the design of our service, followed by the thought behind each of its parts. Lastly we present the process of the implementation for each part of the service.

**Chapter 4** - In this chapter we summarise the outcome of the project, i.e. the resulting prototype, and compare the project goal with the result.
Chapter 5 - In this chapter we give a brief summary of our project. We reflect on our management of our project and which changes we would apply to improve upon it. We finish with a discussion on future work on our project.
Chapter 2 - Background

In this chapter, we introduce WCF Services and some of its benefits, Enterprise Library Logging Application Block, what it does and how it works. We explain the test client and how it was used. We also introduce Silverlight, and XAML. Lastly, we make a short introduction to QlikView, an application for data presentation, before presenting the development tools we have used.

2.1 - WCF Service

In comparison to other services, different kinds of applications can connect to the same WCF service, which is why Prevas wanted us to do our project as a WCF service.

MSDN states that; “WCF enables you to create service oriented applications. Service-oriented architecture (SOA) is the reliance on Web services to send and receive data. The services have the general advantage of being loosely-coupled instead of hard-coded from one application to another” [1].

This means that instead of hardcoding a connection on which the clients and the service communicate the service is able to handle a multitude of different connections as long as the clients meet the essential contracts of the service.

Also according to MSDN [1], a WCF service has several built-in protocols and encodings for transport on which messages can be sent. WCF is able to send SOAP [6] messages, encoded as text or using an optimized binary format, not only using the HTTP protocol but also over TCP named pipes or MSMQ [7]. Using the MOTM [8], standard binary data can be sent efficiently and if needed you can even create your own custom transports and encodings.

For example, consider two different applications: a Java web application and a Windows Forms application. The Java application requires messages formatted in XML format and the protocol to be HTTP, while the Windows Forms application wants messages in Binary format and the protocol to be TCP. If we do not use a WCF service we would have to implement two separate services. See Figures 2.1 and 2.2.
While a client built on any platform can consume a web service, with a remoting service both the client and server must be .NET.

The Web Service and the Remoting Service are two different technologies, and have completely different programming models meaning the developers have to learn different technologies. To unify all technologies under the same umbrella, Microsoft™ offers a programming model called WCF (Windows Communication Foundation).

Instead of creating a .Net remoting service and a .Net web service, only one WCF service is needed as shown in Figure 2.3.

So, it is possible to implement one service which uses multiple endpoints [10]. Endpoints provide clients with access to the functionality that a WCF service offers. You can configure how many endpoints you want to support all your clients’ needs.
Figure 2.3: A WCF service version for the client applications in Figures 2.1 and 2.2 [9].

Our WCF-service should be able to log user statistical information, application performance information and user behaviour information into an XML file. There are some different ways that this can be done of which we have tried two, an XmlWriter and Enterprise Library Logging Application Block.

We started implementing an XmlWriter for our logging functionality. However, Prevas suggested that we should work with Enterprise Library because they thought its Logging Application Block was an easy to use tool for logging, and in the end we found that to be true. The Enterprise Library Logging Application Block is a very effective tool that is easy to configure and work with, see section 2.2.

The collection and structuring of the data is performed on the client side and then sent to our service. The service logs that data with the help of Enterprise Library Logging Application Block.

2.2 - Enterprise Library Logging Application Block
What the logging block does is to provide an easy way of adding a logging functionality for your application outside of your application code, i.e., you get the functionality without writing the code for it.
As stated on MSDN:

The Logging Application Block allows you to decouple your logging functionality from your application code. The block can route log entries to a Windows Event Log, a database, or a text (or XML) file. It can also generate an email message containing the logging information, a message you can route through Windows Message Queuing (using a distributor service provided with the block). And, if none of these built-in capabilities meets your requirements, you can create a provider that sends the log entry to any other custom location or executes some other action [3].

This means that a complex logging of an application can easily be set up by just calling a method of a logging object from the Logging Application Block, such as the LogWriter class, to write the application information to the log. You can configure which accessible data should be logged, in what way and where, see Figure 2.4.

![Diagram of the logging process in the Logging Application Block](image)

**Figure 2.4:** An overview of the logging process in the Logging Application Block [3].
To configure the logger you can either edit the XML configuration file, Web.config, or use the GUI (see Figure 2.5), where you can configure your different categories to use different listeners depending on what type of format you want to save your data to.

**Figure 2.5**: A view of our logging configuration.

In the Categories menu you can add and configure any category you wish to use. The name is used as the identifier of that category when calling the write function of the logger.

A listener is where you configure how the logger handles storing of the log data, i.e., writing to files or databases. For files, as in our case, the listener is a Rolling flat file listener with configurations for where the file should be saved and with what filename, when to create a new file (roll interval), as well as how long these log files should be kept, i.e., automatically deleted after a set number of days. Not all listeners use formatters but our listener is one of those who does use a formatter.
A formatter is, as the name suggests, a configuration setting of the format of the log. We basically have all our collected application data in the *message* of the text formatter. Our formatter also has some extra info about the machine and the application domain, i.e., the server and the server address respectively. In the log message formatter you configure what kind of data that should be saved into the file.

A logger can have many categories which all can have the same log entry assigned to them. Each category can use many listeners, different categories can also use the same listener and different listeners can use the same formatter.

When calling the write method of the logger we pass it two parameters, “xmlString” - the message we wish to log and “category” - the logging category the logger should use. See Figure 2.6.

```csharp
public StatisticsError writeToLog(string xmlString, string category)
{
    try
    {
        Logger.Write(xmlString, category);
    }
    catch (Exception e)
    {
        throw e;
    }
    return null;
}
```

*Figure 2.6: Code snippet of our logging method.*

2.3 - The Test Client
The test client calls the methods in our service asynchronously so that the test client does not have to wait for the service to finish logging. We used this test client to see how our WCF service could handle the different logging operations that we had implemented. See Figure 2.7.
Figure 2.7: The test client application that was used for testing the WCF-service.

**Login** - With the login button we simulate a login by disabling the textbox to the left of it, which should contain the username, and setting the rest of the application to be enabled. When the login button is pressed the client collects performance information about the login operation and statistical information about the user, e.g., which user logged in at what time. The information collected is then sent to the service which saves it in XML format in a log file.

The user information is highly important since if there is ever a system performance issue for some user, it is important to know which user that is. Comparing the performance between that user and others will show if it is an actual issue and what the cause of the issue may be, or if it is a perceived issue.

**Button** - This is our “Search” button which generates some search results based on the text in the textbox below the button and the criteria selected from the list box. When pressed the client simultaneously performs the search itself and collects performance information of the search operation as well as behaviour information of the user, e.g., how many criteria were used in the search and what those criteria were. The information collected is then sent to the service.
The leftmost list box, seen at the top middle of the test client GUI, contains all available parameters to search on. The list box to the top right contains the selected parameters for the search, see Figure 2.8.

![Image of the test client application](image_url)

**Figure 2.8:** The test client application after execution of a simulated login and a search.

The performance information contains the date and time of the operation, how long the operation took to complete as well as which user that triggered the operation.

The statistical information contains, at this point, which user that logged in and at what date and time. Later the date and time of the logout, or closing of the application/connection, will be added to the statistical information before it is logged to file.

2.3.1 - Silverlight

Silverlight [4] is a cross-browser application and a cross-platform technology, i.e. it supports multiple browsers and operating systems. Silverlight is a Microsoft software technology that can be used to display graphics, movies and sounds in a browser. You can also configure your Silverlight application to be run locally instead of on the web.

When first creating a Silverlight project in Visual Studio the program will generate XAML code (Extensible Application Markup Language, see section 2.3.2).
2.3.2 - XAML

As stated on MSDN:

“XAML is the language behind the visual presentation of an application, just as HTML is the language behind the visual presentation of a Web page” [51].

When adding more items to the Silverlight application in the designer view, like buttons and text boxes, more XAML code will be generated. In the Silverlight project, Visual Studio will also generate a C# or Visual Basic file, XAML.cs or XAML.vb, depending on if you are working in C# or Visual Basic. These files are the ones in which the code for the functionality of the buttons and text fields will be implemented.

For an example of XAML for the test client see Figure 2.9.

![XAML code example]

**Figure 2.9:** Some of the XAML generated as the GUI is graphically constructed above.
2.4 - QlikView
QlikView is a Business Discovery platform. It is a program that a company can use to easily gather and show necessary data into straightforward and understandable charts. These charts are then interactive in a way that a user can select a portion of the data pool straight from the charts. All other connected charts update with data for the selected portion, all with a single click of a button.

QlikView can gather data from a variety of different files and formats. QlikView also enables you to do any number of follow-up questions based on what the user has learned, which make this an easy and effective tool to work with [11].

2.5 - Development tools
We developed our WCF service in C# using Microsoft™ Visual Studio 2012 with Enterprise Library 6.0 and its Application Logging Block. The latter can be installed in Visual Studio via NuGet package manager and should be installed in the WCF web Implementation part of the service. We have also used XAML when adding functionality to the test client.

2.6 - Chapter summary
In this chapter we have explained what is needed to know before you could understand the project itself and the implementation we have done. We also gave an introduction of the software tools we have been using throughout this project and explain why we are using them.
Chapter 3 - System prototype design and implementation

In this chapter we explain how we have modelled and implemented our service, starting with its requirements and the thought process behind the information classes and their properties. We then move on to an overview of the design of our service, followed by the thought behind each of its parts. Lastly we present the process of the implementation for each part of the service.

3.1 - Requirements

The requirements of our service are as follows:

- Our service should be implemented in C# and be able to receive and log client usage data.
- The data should be useful for further development of the client applications and for developing new applications.
- The data should be logged in a way that it is readable by QlikView in which the statistics of the data should be presented.

The suggestions of data that could be useful for improvement of applications were:

- Which operation that was executed.
  - When it was executed.
  - By whom it was executed.
  - The time span needed for the execution to complete.
  - If the operation was a search:
    - How many criteria that were used in the search.
    - What the criteria were.
    - How many hits the search got.
    - How many data fields that were presented from the search.
- The user.
  - How often the user logs in.
  - How often the user logs out.
  - How long the user is logged in.
  - How many users that are logged in simultaneously.
We needed to sort the data into categories or classes, user statistics, performance and user behaviour.

Some of the data were not necessary in all classes. Some of the data was implicit, i.e. it could be gathered from combining two or more pieces of data, and as such not necessary to log.

The resulting categories and their data:

- **User Statistic (Login/Logout):**
  - Time of log in.
  - Time of log out.

- **Performance**
  - Name of operation.
  - Time of operation.
  - User of operation.
  - Duration of execution.
  - Operation specifics:
    - Resulting hits of search operation.
    - Resulting fields of search operation.

- **User Behaviour**
  - Name of operation.
  - User of operation.
  - Operation specifics:
    - List of criteria used.

These properties were then implemented in the data classes in our interface layer. Each category represents a class and each of its data a property of that class.

To be able to design and implement the service we were first required to know how WCF services work and how to implement such a service. We also needed to know how to use and configure Enterprise Library Logging Application Block.
3.2 - Design

Our project is a prototype of a service module, that collects usage statistics, consisting of four layers:

- Interface layer (Contracts).
- Data access layer (DAL).
- Business logic layer (BLL).
- WCF web implementation (WEB).

These four parts are useful in completely different ways to the project. We structured this application so that you could easily change any part you would want to reconfigure, or even swap it out completely. This is the main reason to why we implemented the service in four different layers, see Figure 3.1.

![Figure 3.1: A map over the structure of our service.](image-url)
3.2.1 - WCF web implementation, WEB
In the top we have the WCF web implementation. This is the layer where all data flows through from the test client to our WCF service. The WEB is the service without any functionality. It is through this layer the clients connect to our service. This is also where we have the endpoints configured. See section 2.2 about WCF service and endpoints. In the WEB we are able to choose where to handle the data after it has been received by our service.

3.2.2 - Business Logic Layer, BLL
In the middle we have the Business Logic Layer. This is the layer where we keep our methods and operations that actually process the data. The layer that has all the functionality the WCF service would need. In this particular case we build a string in XML format of the information we gathered and send this string to our next layer, the Data Access Layer (DAL). If we should ever want to change operations in the future we can simply just add the method or swap the whole layer if we would want another kind of setup for this layer.

3.2.3 - Data Access Layer, DAL
Then we have the Data Access Layer. This is where the decision the information is going to be saved into a database or just into a single file is made. This is the layer where any method that accesses a file or database is kept. Since we use Enterprise Library Logging Block, we call upon the logger’s write method to write to the log file(s). If we ever want to change where the information is saved we could easily just swap the data access layer.

3.2.4 - Interface layer, Contracts
This is where we have our interface class, IStatisticsService, and any other class that should be known to all parts of the service, like the different data classes (PerformanceInformation, UserBehaviourInformation and UserStatisticInformation). This layer also has a class for handling errors the service may throw. If we get an error we will return an error message and the code for that message.

To summarize, we have the WEB at the top which has references to the BLL and the Contracts. The DAL lies at the bottom also has reference to the Contracts and handles export
and import of data to and from a database or file. In between these we have the BLL with references to the DAL and the Contracts, see Figure 3.1.

The Contracts (the interface layer) is referenced by all the other layers so that they know of the data types.

The reason behind creating these different layers is to gain modularity within the service so that one layer can be swapped out with another implementation of the same layer.

3.3 - Implementation
Our service is coded in the C# language in the Microsoft ™ Visual Studio [12] environment, along with the use of Enterprise Library. For the modifications of the Silverlight test client we have used XAML along with the graphical designer view in Visual Studio.

We implemented a Visual Studio solution with a WCF service application and three class libraries. The class libraries are the BLL, the DAL and the Contracts, leaving the WEB which is the WCF service application. All references were then added as per the design (see Figure 3.1), and the XML-files for cross domain and client access policy were added to the WEB layer according to MSDN directions [13], with asterisks a.k.a. wildcards (*) as header attribute values (as seen in the comment by Stefan U7, posted 10/12/2012 also see section 3.3.1).

We chose to implement our service in these separated class libraries so that we could easily swap a class library out later on, if ever needed. This is a structure that Prevas usually works with so we chose to implement it in the same way. When we first started this project we chose to do this project using C# since we had some earlier experience with this language and since it was a request by Prevas.

In the Silverlight test client, that Prevas provided, we added a login simulation functionality, where we could type in a user name and “log in”. We also set all other components in the test client GUI to be disabled from start and to only be enabled while a user was simulated to be logged in, this forced a login before any other action just like a real client.
3.3.1 - WCF web implementation, WEB

The WEB layer contains both implementation and configuration. In this layer we have the WCF implementation which inherits the interface from the interface layer and implements its methods.

We first instantiate a BLL so we get a connection between the BLL layer and WEB layer, see Figure 3.2.

```csharp
public class StatisticsService : IStatisticsService
{
    StatisticsLogic bll = new StatisticsLogic();
    /// <summary> ... 
    public StatisticsError SavePerformanceInformation(
        Contracts.Custom.PerformanceInformation info)
    {
        return bll.SavePerformanceInformation(info);
    }
    /// <summary> ...
    public StatisticsError SaveUserBehaviourInformation(
    {
        return bll.SaveUserBehaviourInformation(info);
    }
    /// <summary> ...
    public StatisticsError SaveUserStatisticInformation(
    {
        return bll.SaveUserStatisticInformation(info);
    }
}
```

**Figure 3.2:** Code snippet of the WCF implementation, StatisticsService.svc.cs.

The implementation of these methods is only to forward the data to the next layer, because of how we have structured the design where the WEB should not be able to do more than receive the data and send it forth.

We have also implemented and configured two XML documents to this layer so that we could have cross domain functionality, see Figures 3.3 and 3.4. These documents were added to the WEB solution.
3.3.2 - Business Logic Layer, BLL

In the Business Logic Layer we implemented several methods for receiving and working with data we gathered from the WEB layer. The methods shown in Figure 3.5 build an XML string from the data we received from the WEB layer and sends that data to the DAL.
We also have a reference to the interface class so the different methods know what kind of information that they will be able to receive and forward. For the complete code of the BLL see Figure A.2 in Appendix A.

3.3.3 - Data Access Layer, DAL
In the Data Access Layer we implement the logging functionality for our WCF service. We first initiate a logger and then we have the received data from the BLL layer as input to the logger. See Figure 3.6.

The configuration of the Enterprise Library Logging Block we have done in the WEB layer dictates how the logger should handle the information depending on which category is sent to the logger.
3.3.4 - Interface layer, Contracts

In the Interface layer we created two folders, “Interface” for the application interface (service contract), and “Custom” for our data types (data contracts). Every operation contract in the service contract has a corresponding data contract. This layer contains all the data contracts and operation contracts constructed for our service, see Figures 3.7 and 3.8.

```csharp
namespace WCFStatistics.DAL
{
    public static class StatisticsAccess
    {
        static StatisticsAccess()
        {
            Logger.SetLogWriter(new LogWriterFactory().Create(), false);
        }
        public static StatisticsError writeToLog(string xmlString, string category)
        {
            try
            {
                Logger.Write(xmlString, category);
            }
            catch (Exception e)
            {
                throw e;
            }
            return null;
        }
    }
}
```

**Figure 3.6:** Code snippet of the class in our DAL layer, StatisticsAccess.cs.

```csharp
[ServiceContract]
public interface IStatisticsService
{
    [OperationContract]
    StatisticsError SavePerformanceInformation(PerformanceInformation info);
    [OperationContract]
    StatisticsError SaveUserBehaviourInformation(UserBehaviourInformation info);
    [OperationContract]
    StatisticsError SaveUserStatisticInformation(UserStatisticInformation info);
}
```

**Figure 3.7:** Code snippet of the interface, IStatisticsService.cs.

In the interface class the service contract is declared with the operation contracts and in the “Custom” folder we have the different data contracts for the different information types, i.e.,
performance, user statistics and user behaviour. The “Custom” folder is also where we have the error type class for our service.

```csharp
[DataContract]
public class UserBehaviourInformation
{
    [DataMember]
    public string user { get; set; }
    [DataMember]
    public string functionName { get; set; }
    [DataMember]
    public List<string> criteria { get; set; }
    [DataMember]
    public string category { get; set; }
}
```

**Figure 3.8:** Code snippet of one of the data contracts, UserBehaviourInformation.cs.

### 3.3.5 - The test client

In the test client we added a button and a text box to the GUI elements. We also implemented the functionality to the button that would simulate a login where the text in the text box is the user name. The code for the login functionality can be seen in Figure A.9.

Besides adding the GUI elements we implemented code to set values of the data classes before sending the data collections to our service. See Figure A.10 for that code in the search method of the test client. The data is sent to our service by asynchronous calls. We also implemented the call-back methods for the asynchronous calls, see Figure A.11.

### 3.4 - Chapter Summary

In this chapter we have gone through the requirements we had on our service and what requirements were needed to be able to design and implement the service. We have shown how we modelled and designed our service and how it was implemented.

The service is built with different layers:
- The WEB layer through which the clients connect and call on the service methods.
- The BLL in which the service process the data.
- The DAL which initiates the logger and calls its write method.
- The Interface layer which holds the interface and data classes.

The results of our project are presented in the next chapter.
Chapter 4 - Results

In this chapter we summarise the outcome of the project, i.e., the resulting prototype, and compare the project goal with the result.

4.1 - The Resulting prototype

We successfully implemented a working WCF service that receives information from a client and builds an XML string, then sends the string to our Enterprise Library logger that generates a log file that contains the XML string. All our produced code for the service can be found in Appendix A.

We were, by manually encapsulating the log entries in a single root element, able to show the information we have gathered from the test client in different charts and tables using QlikView, see Figure 4.1 and 4.2.

![Figure 4.1: Example view of data presentation in QlikView.](image)
The data used for these charts were generated by placing the functionality of the test clients search button in a for loop, generating random timestamps and then letting QlikView generate usernames by using modular arithmetic on the number of log entries made. The data presented is therefore not representative of actual usage of a service, it is however useful to show what can be presented from the data which our service logs.

In Figure 4.1 the data has been limited down to the year and month selected. The top chart presents average response time and number of returned rows, i.e., search hits, for users. The response time is in milliseconds and shown on the left y-axis while the returned rows is shown on the right y-axis.

The middle chart presents average response time and number of returned rows depending on the number of search criteria used. The response time, in milliseconds, is shown on the left y-axis and number of returned rows on the right y-axis. The bottom chart presents the average number of criteria used for each user.

![Figure 4.1](image)

**Figure 4.1:** Example view of data presentation in QlikView.

In Figure 4.2 the data has been limited down to the year, month and day selected. The top charts present the average number of search criteria used per group of users and per user.
respectively. The bottom charts present the number of searches made per group of users and per user respectively.

4.2 - Project goal vs. result
The expected results of this project were a finished service prototype that could log any action taken in a test client. The data collected should tell us which user who triggered which operation and the performance of that specific operation. The service should then convert and store that information into an XML-file. The data in the XML-file should then be presented in a program called QlikView.

Our actual result is as follows: We successfully implemented a WCF service that can convert and store received information in a log file containing the data in XML format. The test client, that Prevas provided us with, was edited and set up with our service and we got our service to log the data using Enterprise Library Logging Application Block. The log file, however, is not quite an XML-file since every log entry is a root element, i.e., there are multiple elements at root level which makes it an invalid XML-file.

This means that the actual result meets the expected result, except for the root element of the log file.
Chapter 5 - Conclusion

In this chapter we give a brief summary of our project. We reflect on our management of our project and which changes we would apply to improve upon it. We finish with a discussion on future work on our project.

5.1 - Summary
Our project was a WCF service that can receive and log data from a software application connected to our service. The data is logged to file using Enterprise Library Logging Application Block. See chapter 3 on the design and implementation of our service.

This project has given insight on how WCF services and Enterprise Library Logging Application Block work and their advantages. See section 2.1 – WCF Service and 2.2 – Enterprise Library Logging Application Block.

Besides performing our literature study of WCF services, we were handed the code of a WCF service that Prevas had to test some features of a test client. It was difficult for us to understand the code of the WCF service we were given as we were not familiar with either WCF services or asynchronous calls which that service made use of. We familiarised ourselves with these aspects through our literature study on WCF services and by experimenting with the code we were given.

5.2 - Project Management
In retrospect there are a few aspects of the project management that could have been done differently. If we were to start over with this project we would put, if not equal then almost equal, focus on the thesis and the development from the start. We believe doing so would help us to structure both our thesis and our project.

We would also use an Agile development technique called Scrum [14], and perhaps even TDD (test driven development) [15], which is a style of programming. Using Scrum would help in structuring the project and managing our progress throughout the timespan of the course, i.e., making sure our project progressed in a steady pace. Using TDD would give quality insurance on our work.
5.3 - Future work / Next step

The next step in our project would be to get EquipMan [16] to interact with our service since EquipMan is the main application from which Prevas wants to get usage statistics and is hence the main application which plans to make use of our service.

EquipMan is an application used by a Nordic tele-company to keep track of all their hardware globally. EquipMan has much more functionality and can generate a lot more information compared to our test client, see figure 5.1.

![Figure 5.1: View of what the search page in EquipMan looks like.](image)

Some methods and data classes would have to be added, and some existing methods and data classes might have to be altered. The data for performance, for instance, might have to be divided into a more general performance for any operation and an operation specific data collection.

The WCF service we have implemented should already be able to work together with EquipMan. However, the information that can be logged would probably not be sufficient enough to use for improvement of more than one aspect of the application, namely the performance of the searches in the application.
References


Appendix A - Code

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using WKFStatistics.Contracts.Interface;
using WKFStatistics.Contracts.Custom;
using WKFStatistics.BLL;

namespace WKFStatistics.WEB
{
    public class StatisticsService : IStatisticsService
    {
        Statisticslogic bll = new Statisticslogic();
        // <summary> ...
        {
            return bll.SavePerformanceInformation(info);
        }
        // <summary> ...
        {
            return bll.SaveUserBehaviourInformation(info);
        }
        // <summary> ...
        {
            return bll.SaveUserStatisticInformation(info);
        }
    }
}
```

**Figure A.1:** StatisticsService.svc.cs (WEB).

This is the code for our WEB where we have three different methods. Each of them sends data to the BLL.
This is our StatisticsLogic class where we build our XML string from the data received by our service. We also check if the category for our Enterprise Library logger is specified, if not it will be the default which is “Combined”.

Figure A.2: StatisticsLogic.cs (BLL).
This is our StatisticsAccess class where we initiate our logger and use the data gathered and the category info as input parameters.
In our Interface folder in our Contracts we have the IStatisticsService class, the interface, which contains operation contracts for the methods our service will use.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using WCFStatistics.Contracts.Custom;

namespace WCFStatistics.Contracts.Interface
{
    [ServiceContract]
    public interface IStatisticsService
    {
        [OperationContract]
        StatisticsError SavePerformanceInformation(PerformanceInformation info);
        [OperationContract]
        StatisticsError SaveUserBehaviourInformation(UserBehaviourInformation info);
        [OperationContract]
        StatisticsError SaveUserStatisticInformation(UserStatisticInformation info);
    }
}
```

**Figure A.4:** IStatisticsService.cs (Contracts).

In our Custom folder in our Contracts we have the UserStatisticInformation class in which we have the properties for the data we use in the SaveUserStatisticInformation method.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Runtime.Serialization;

namespace WCFStatistics.Contracts.Custom
{
    [DataContract]
    public class UserStatisticInformation
    {
        [DataMember]
        public string user { get; set; }
        [DataMember]
        public DateTime logInTime { get; set; }
        [DataMember]
        public DateTime logOutTime { get; set; }
        [DataMember]
        public string category { get; set; }
    }
}
```

**Figure A.5:** UserStatisticInformation.cs (Contracts).
In our Custom folder in our Contracts we have the UserBehaviorInformation class in which we have the properties for the data we use in the SaveUserBehaviorInformation method.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Runtime.Serialization;

namespace WCFStatistics.Contracts.Custom
{
    [DataContract]
    public class UserBehaviourInformation
    {
        [DataMember]
        public string user { get; set; }
        [DataMember]
        public string functionName { get; set; }
        [DataMember]
        public List<string> criteria { get; set; }
        [DataMember]
        public string category { get; set; }
    }
}
```
In our Custom folder in our Contracts we have the PerformanceInformation class in which we have the properties for the data we use in the SavePerformanceInformation method.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Runtime.Serialization;

namespace WCFStatistics.Contracts.Custom
{
    [DataContract]
    public class PerformanceInformation
    {
        [DataMember]
        public string functionName { get; set; }
        [DataMember]
        public DateTime occurrence { get; set; }
        [DataMember]
        public string user { get; set; }
        [DataMember]
        public TimeSpan totalTime { get; set; }
        [DataMember]
        public UInt32? numberRows { get; set; }
        [DataMember]
        public UInt32? numberCols { get; set; }
        [DataMember]
        public UInt32? numberValues { get; set; }
        [DataMember]
        public string category { get; set; }
    }
}
```

**Figure A.7**: PerformanceInformation.cs (Contracts).
In our Custom folder in our Contracts we have the StatisticsError class. This is the class we use when returning errors in our service.
Figure A.9: The login functionality we implemented in the test client.
Figure A.10: The collecting of data that we implemented in the test clients search method.

Our code is the code which is in between the #region and #endregion markers. What is above
the #region marker in this figure is data that were already available to use when Prevas supplied us with the test client.
Figure A.11: The asynchronous call-backs we implemented in the test client.
Appendix B - Solution chart

![Diagram of Visual Studio solution for service]

**Figure B.1**: A chart over our Visual Studio solution for our service.