Distributed logging server

Andrei-Alexandru Agape

This report presents the architecture, the technologies and the steps needed in order to develop and implement a distributed logging server.

Company: Accenture, Romania

University: UCN Aalborg, Denmark

Supervisor: Simon Kongshø

12/22/2014

Table of Contents

1. Introduction

1.1 Problem statement

One common logging problem of the applications (distributed or not) is that as the time passes and they become bigger, their logs grow also: in size, in details and most important in quantity. The fast and easy solution when the amount of data saved in one file is way too big is to split the data in more files. But this is just a temporary solution and sooner or later there will be a lot of files and reading the logs will involve many of them. Logs that are related one to another, but are sliced in many files are hard to follow and understand.

Saving the log details into files, as a very long string is another common habit of the classic logging systems. This can be useful and easy to debug when the number of logs is low, and their details does not include too many information despite a time stamp and a message. But when the logs are very detailed and their number is higher, the log files will start to look like a block of text hard even to read, barely to understand anything. Beside the fact that they are hard to read, the log files are hard to process and filter their information without additional work. Read line by line, split the string into columns, convert each column to its type are just a few steps that have to be done so that we can use an automatic search on this information. And even so we do not have the guarantee that everything will work because the data we receive can be in an unexpected format.

Usually when a client runs in to an error or a problem of the application, he/she tries to solve it by following the error messages, and if there are not enough details simply ignores it. Sometimes though the developer is announced regarding the problem, but unfortunately the error can seldom be reproduced or any additional details presented. Thus, the developer has to search through a block of text inside the log files for an error about which he does not know barely anything. The probability to solve the problem may not be so low if he would be able to find and read the log details for the specific problem.

All these presented above, alongside with the fact that the log files can be easily lost, hard to keep a backup for them and the lack of security, present the main problems that the classis system of logging has even in medium size applications. In a distributed application, to all these we can add the log transmission from one server to another and also the security that has to be ensured when we do this.

A short summary of the ideas that were discussed this chapter and the problems identified is

- Avoid to save logs in files at the risk of loose them or be read by unauthorized people
- Difficulties in filtering and reading the logs due to the fact that are saved in .txt files
- Difficulties in solving the errors due to the fact that the clients cannot reproduce them or not enough details are given
- Decentralized logs in distributed applications are hard to read and follow

1.2 Main goals

There is a moment in the life of every developer when he/she cannot understand the logic behind the returned result or the application behavior, thus the fastest way to debug this is by printing a message or a variable to the console to see if the values are the same as he/she expects them to be. That message outputted is nothing else but a log which traces the evolution of the application and displays details about the logic flow, helping him/her to understand and solve the bug.

This may be easy to do and useful in small applications, but the number of log messages grows proportionally with the number of lines of code and sooner or later it will be impossible to read through the logs. Actually, there won't be a console to write and read from. And just a simple message may not be enough; things like stack trace, user name, function name, time stamp etc. are necessary and we may want to filter these by their importance (eg.: error, info).

But if logging inside a regular application may become hardly possible without a special tool – a logging framework - what kind of tool can be used to log inside a **distributed** application?

Answer: A distributed logging server.

If we have a large or distributed environment that requires logging from many instances, we have 2 main options:

- Configure all the instances to independently record their data in the same database.
- Configure all the instances to send their logs to a distributed logging server, which then records all the information to the database.

The first option does not require special configuration steps. We need only to ensure that each instance points to the same database (both database engine IP address or host name and database instance name).

In the second option it may look that we have one extra layer in our logging process, the logging server, but by implementing it the complexity of adding new instances to server will be constant. Another thing to take into consideration are the changes that may occur in the database location/architecture/etc. – having a server which deals with all those details we only have to change its configuration and not for all the instances. Thus, we do not have to worry about the fact that not all the instances are configured correctly or because of the duplicated configuration files.

Once we understood the advantages of using a server to collect the logs from all the instances, it is obviously that this is also the best solution in case of a distributed application which may have tens or hundreds of instances on many servers.

Another very important thing that the server should take into consideration is the ease of implementation. Since not all the applications may have a distributed logging server implemented from the beginning, and some of them will decide to implement one only after a few months after a stable version of the application has been released, it is important for the server to be easy to implement, easy configurable and does not affect the structure and logic of the existing code. The logs are meant to help the developer to understand the application’s flow and its bugs and problems – having a server that is hard to implement and creates even more bugs in the application will make the situation ridiculous.

Not all the applications are released without a logging system – in fact many of them have one that was very useful in the development phase but once the application is sent in the production that system becomes obsolete and old simply because the developers cannot anticipate all the scenarios that the user will cause. Their logging system may have been useful in well-known scenarios, with a limited amount of actions and in a testing environment, but in a real world scenarios where many users cause hundreds or thousands of logs in a very short interval of time, it won’t take too much until it will be impossible to read and debug through the basic logging system.

This is the main reason of why the server should not only be easy to implement, but also can easy substitute existing logging systems that cannot deal anymore with the amount of data. And by easy substituting I mean that the amount of changes in the already written code has to be minimum or even none, while the log processing inside the logging server has to be the one that does most of the work.

As it is known, a distributed application can have instances on many servers and when it comes to communication between these instances that are spread on many servers there is of course the probability that the connection between them can be lost. Losing the connection between the instances that write the logs and the logging server means that all those logs are not recorded anywhere, thus they are lost. These kind of situations is unwanted, even if the connection is lost just for a few seconds, the information inside the log database will be wrong since some steps in the application flow are missing.

In order to anticipate this kind of situations and to solve them elegantly, implementing a fallback solution for these cases would be very useful. The fallback solution should be able to know when the connection between the instances and server is not possible and then automatically switch to another target (eg.: saving the logs locally). Once the connection is reestablished, the server should receive all logs that were stored locally by the instances and process them as normally. All this process has to be done automatically, so that by checking the logs there won’t be any differences between the logs done when to connection was available and when it was not.

But what is the use of the logs if it takes a long time to read and understand them, if we have to open tens of files and follow the logic from one to another? Debugging the logs in order to debug the errors may be even more annoying. The difference between having each instance logging for itself or having just one server that process the logs received from instances is the centralized information. And this is another main goal of the server – to centralize the information and access all the instances logs from a single place like a control panel. All the logs from all the instances have to be stored in a single place, either a file, a database or table storage on cloud.

Usually developers who read through logs are humans and despite the amount and quality of information that the logs contain, a big impact has the way in which those logs are displayed. Display the logs as a long string inside a .txt file and display them inside a database. Even if the logs are the same, reading from a well-structured database will be way much easier not only to read, but also to understand the information. Of course, as its name says, a database is meant only to store the data even if some applications allows us to display the information, those are most of the time not enough and not suitable. A well personalized GUI Client that allows the developer to display the logs stored would be much appreciated. This client should be able to filter through logs, display them in an easy-to-read format and update in real-time with the logs that are made.

Additional to this client features, downloading the logs to a file offers the possibility to be independent by the client and have a snapshot of a particular error, alongside with its context (eg.: the last hour actions that lead to an event).

Most likely there won't be a developer dedicated to only monitories the logging "control panel", but only check it times to times or when an error is reported by a client. Due to high amount of data and the fact that the clients may not always be able to reproduce the error, an useful help for the developer would be that the client download the logs for his error and sends it further to the developer. This does not require for the client to reproduce the error, while the developer can save time in the process of searching for that particular error. This of course has to take in to consideration some security issues that may occur and that are discussed later in the report.

A short summary of the ideas that were discussed this chapter is:

- To have a server that collects the logs from many distributed applications
- To easy implement the server without affecting the existing code
- To easy replace the existing way of logging
- To save the logs even when the server is down or the connection is not available
- To search through all the logs in a single place regarding the connected applications
- To download the logs to file
- To have a Graphic User Interface that display the logs and filter by criteria
- To have a Friendly GUI for the clients that will encounter errors and allow them to download the log for their problem

1.3 Similar projects

"A new class of solutions that have come about have been designed for high-volume and high-throughput log and event collection. Most of these solutions are more general purpose event streaming and processing systems and logging is just one use case that can be solved using them. All of these have their specific features and differences but their architectures are fairly similar. They generally consist of logging clients and/or agents on each specific host. The agents forward logs to a cluster of collectors which in turn forward the messages to a scalable storage tier. The idea is that the collection tier is horizontally scalable to grow with the increase number of logging hosts and messages. Similarly, the storage tier is also intended to scale horizontally to grow with increased volume. This is gross simplification of all of these tools but they are a step beyond traditional syslog options."

Similar distributed logging servers exists, through the most popular we can mention:

- **1.3.1 Scribe** - Scribe is scalable and reliable log aggregation server used and released by Facebook as open source.
- **1.3.2 Flume** - Flume is an Apache project for collecting, aggregating, and moving large amounts of log data.
- **1.3.3 Kafka -** Kafka was developed at LinkedIn for their activity stream processing and is now an Apache incubator project. Although Kafka could be used for log collection this is not it's primary use case.

Even if all these servers are great tools, not all of them may satisfy the specific needs for an application so writing our own distributed log server is not uncommon taking in consideration the fact that is:

- **Easy to implement** - They generally consist of logging clients and/or agents on each specific host. The agents forward logs to a cluster of collectors which in turn forward the messages to a scalable storage tier.
- **More configurable** – By writing our own logging server we are free to customize and configure it exactly as we want, without any restrictions: details that are recorded by the logs, the way they are sent and the real-time distribution.
- **Cheaper** – Designing your own logging server may be cheaper on long term, even if there is a lack of features like technical support or periodically updates, those depends only on us.
- **Fully customizable on all levels** – As any other product, an already implemented logging server will come with some configurable options as well as some out of box on which we have no power to change. Those are not valid on a product made by ourselves.
- **Not dependable on others for future versions** – whenever an error occurs, or a new feature is needed, we can simply implement/solve it without having to wait for an update of the product.

2. Technology choices

2.1 Logging Framework

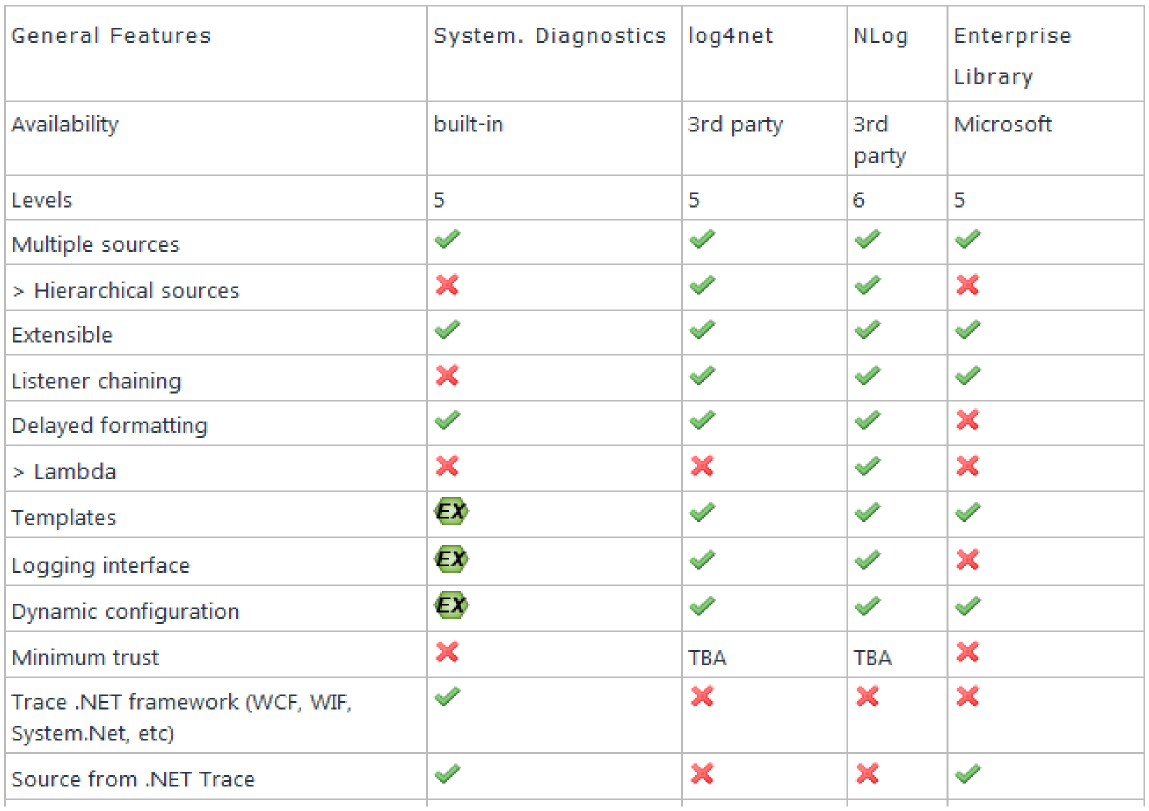
Even if some solutions for a distributed logging server already exist, making our own server has of course some advantage like: easy configurable, fully customizable, not dependable on other releases and even the price. But when it comes to create a distributed logging server, the main part of it is of course the logging system itself and even if it would be possible to create our own methods and functions for this, choosing a logging framework is the wise decision in this case. Logging frameworks exists for many years, useful tools that deals with many problems that we might not even think about: support for many targets, rolling files or trace context are just a few of them.

After a short research on the logging framework market, the best and most common solutions are:

- Log4net
- NLog
- Enterprise Library

Bellow can be found tables that present a comparison between those frameworks taking in consideration factors like:

2.1.1 Availability



General Features	System. Diagnostics	log4net	NLog	Enterprise Library
Availability	built-in	3rd party	3rd party	Microsoft
Levels	5	5	6	5
Multiple sources	✓	✓	✓	✓
> Hierarchical sources	✗	✓	✓	✗
Extensible	✓	✓	✓	✓
Listener chaining	✗	✓	✓	✓
Delayed formatting	✓	✓	✓	✗
> Lambda	✗	✗	✓	✗
Templates	EX	✓	✓	✓
Logging interface	EX	✓	✓	✗
Dynamic configuration	EX	✓	✓	✓
Minimum trust	✗	TBA	TBA	✗
Trace .NET framework (WCF, WIF, System.Net, etc)	✓	✗	✗	✗
Source from .NET Trace	✓	✗	✗	✓

2.1.2 Filters

Filters	System. Diagnostics	Log4net	NLog	Enterprise Library
Event level	✓	✓	✓	✓
Source	✓	✓	✓	✓
Property	EX	✓	✗	✗
String match	✗	✓	✓	✗
Expression	EX	✗	✓	✗
Priority	✗	✗	✗	✓

2.1.3 Log Information

Log Information	System. Diagnostics	Log4net	NLog	Enterprise Library
Event ID	✓	contrib extension	✗	✓
Priority	✗	✗	✗	✓
Process/thread information	✓	✓	✓	✓
ASP.NET information	EX	✓	✓	✓
Trace Context	✓	✓	✓	✓
> Correlation identifier	✓	✗	✗	✓
> Cross-process correlation	✓	✗	✗	✓
Exceptions	✗	✓	✓	via exception block

2.1.4 Listeners

Listeners	System. Diagnostics	Log4net	NLog	Enterprise Library
> Forward to .NET Trace	✓	limited	limited	✓
ASP.NET Trace	✓	✓	✓	✗
Chainsaw (log4j)	✗	✓	✓	✗
Colored Console	EX	✓	✗	✗
Console	✓	✓	✓	✗
Database	EX	✓	✓	✓
Debug	✓	✓	✓	✗
Event Log	✓	✓	✓	✓
Event Tracing (ETW)	✓	✗	✗	✗
File	✓	✓	✓	✓
Mail	EX	✓	✓	✓
Memory	EX	✓	✓	✗
MSMQ	✗	✗	✓	✓
Net Send	✗	✓	✗	✗
Remoting	✗	✓	✗	✗
Rolling File	✓	✓	✓	✓
Syslog (unix)	✗	✓	✗	✗
Telnet	✗	✓	✗	✗
UDP	UdpPocketTrace extension	✓	✓	✗
WMI	✗	✗	✗	✓
XML (Service Trace)	✓	✗	✗	✓

This comparison makes it easier to reduce the number of possible solution to only 2: **NLog** and **Log4Net**. A comparison was then made between those 2 candidates in order to decide which one fits better our distributed logging server needs.

In order to do a comparison between them, an important role had all the developer's opinions that use these frameworks before and who left feedback regarding their pros and cons. Trough the most important factors remembered by everyone were:

2.1.5 How often the frameworks are updated and how often new versions are released?

NLog is a more active project, releasing new versions and updates once every 6 months currently newest version being released in July 2014. Even if Log4net had also release a new version not long ago (2013), it was important to see that Log4net was almost inactive for a few years since (2006 – circa 2011).

2.1.6 How easy is to implement and use?

"*When revisiting things a few weeks later, NLog was* clearly *the easiest to resume. I needed very little brush up on it. With Log4Net, I had to revisit a few online examples to get going*."

"*NLog has a config file schema so you get "intellisense". Log4Net configuration has no support*."

"In log4net, internal logging does not output timestamp which is annoying. In Nlog, you get a nice log with timestamps. I found it very useful in my evaluations."

"Setting/starting up with NLog is dead easy. You go through the Getting started tutorial on their website and you are done. You get a fair idea, how thing might be with nlog. Config file is so intuitive that anyone can understand the config. For example: if you want to set the internal logging on, you set the flag in Nlog config file's header node, which is where you would expect it to be. In log4net, you set different flags in web.config's/ appSettings section."

2.1.7 What is the performance and what are their features ?

"Log4net does not support asynchronous appender."

"Performance - I logged around 3000 log messages to database using a stored procedure. I used simple for loop (int i=0; i<3000; i++). To log the same message 3000 times. For the writes: log4net AdoAppender took almost double the time than NLog."

Log4net does not support MSMQ target which was meant to use as a fallback solution for the distributed logging server.

All these opinions, statistics and the personal tests of the frameworks lead to the idea that NLog would be the better solution in the scope of developing a distributed logging server.

2.2 Data storage

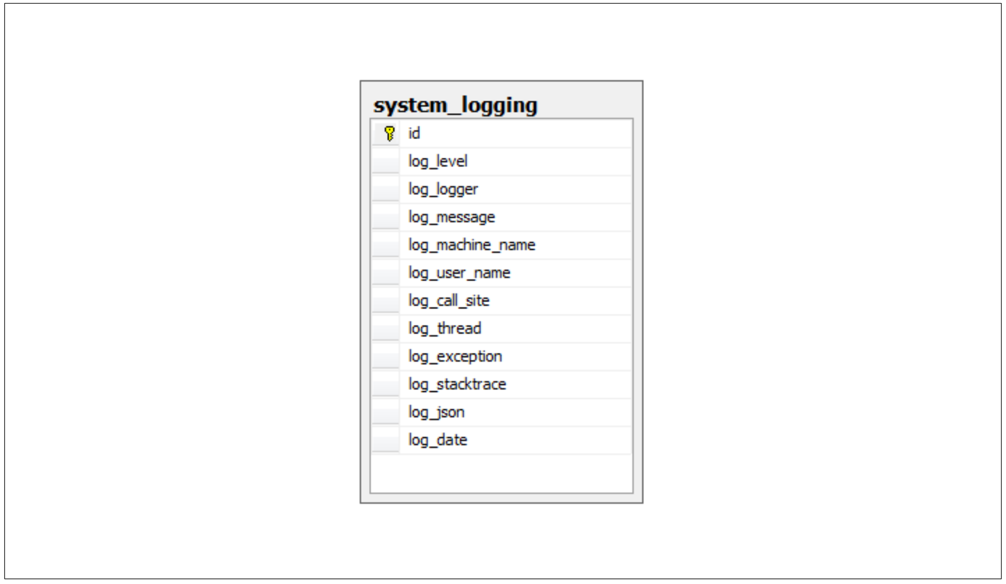
Once the decision of using NLog as logging framework was made, the next thing to decide was the storage location of the logs. The main options were the database, the cloud and the files.

Since one of the main problems mentioned in the problem statement was the fact that is barely possible to read and debug the logs from the files, the file option was not suitable for permanent storage but still being useful for temporary cases and for 'snapshots' of the data being easier to transport a small amount of data.

A deep comparison between the database option and the cloud was not made, but only simple tests that showed no big differences at all. Since the NLog syntax for the Database target and for the Azure Table storage is almost the same, the only factor that make the target to be a Database was the fact that sometime the server may not be connected to the internet, thus a local database would be more suitable.

The database structure was straight-forward, with only one table containing all the details a log will present as columns. Beside the **ID**, **Log_Thread** and **Log_Json** which are set to be integers, all the other columns are meant to be strings.

Since the logging process can be very intensive in production environment, a possibility for the ID would be to use a GUID (**Globally Unique Identifier**) because the int32 is limited to 4 million unique possibilities while GUID "has the probability of one duplicate would be about 50% if every person on earth as of 2014 owned 600 million GUIDs". But since this was a proof of concept the type used was not so important, but it was good to be mentioned and take into consideration for real distributed logging systems that will be developed in the future.



system_logging

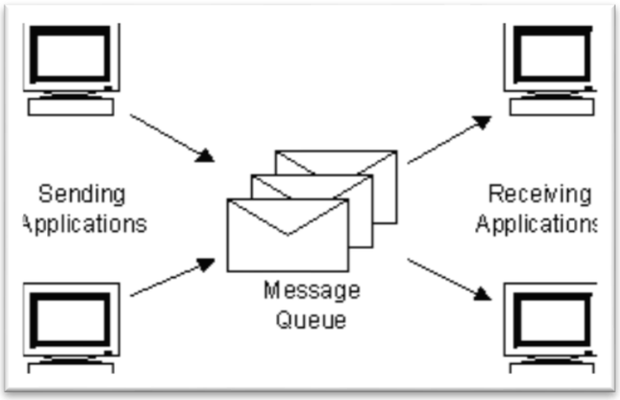
id
log_level
log_logger
log_message
log_machine_name
log_user_name
log_call_site
log_thread
log_exception
log_stacktrace
log_json
log_date

2.3 Fallback options

MSMQ

Another important factor in the distributed logging server architecture is the fallback option for the situations when the connection may not be available. This factor had a big influence in choosing also the logging framework because, as is specified earlier, the Log4net framework does not support MSMQ as target.

Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues. The following illustration shows how a queue can hold messages that are generated by multiple sending applications and read by multiple receiving applications.



As the Microsoft Documentation specifies, the MSMQ is suitable Message Queuing provides guaranteed message delivery, efficient routing, security, and priority-based messaging.

It can be used to implement solutions to both asynchronous and synchronous scenarios requiring high performance.

The MSMQ fits perfectly the scenario when the logging server is too busy to process all the requests or it is offline for an undetermined period of time.

2.4 Real-time data transmission

In some applications a critical error may cause loss of data, security issues or failure of system for a period of time. Sometimes, those critical errors can be avoided if we anticipate and recognize them in the log messages (eg.: many errors in a short interval of time, number of users growing unexpected etc.)

In order to be able to anticipate the failures and read these signals, the transmission of logs has to be done real-time. Reading the logs after the critical failure has already happened may help in understanding the problem and solving it so it won't happen in the future, but it does not help to avoid the current failure. To be able to transmit the data in real time, it is needed that the server will push the logs not only in the database, but also to the logging admin viewer which will display them immediately.

Two main technologies are used to transmit the data in real time: SignalR and Ajax.

2.4.1 SignalR

ASP.NET SignalR is a new library for ASP.NET developers that makes it incredibly simple to add real-time web functionality to your applications. What is "real-time web" functionality? It's the ability to have your server-side code push content to the connected clients as it happens, in real-time.

SignalR also provides a very simple, high-level API for doing server to client RPC (call JavaScript functions in your clients' browsers from server-side .NET code) in your ASP.NET application, as well as adding useful hooks for connection management, e.g. connect/disconnect events, grouping connections, authorization.

Thus, SignalR is the best solution when it comes about the complexity of implementation, the technology perspective in the future and the ease of development. In the distributed logging server, SignalR is used to push the logs further to the logging admin viewer.

2.4.2 Ajax

AJAX = Asynchronous JavaScript and XML.

In short; AJAX is about loading data in the background and display it on the webpage, without reloading the whole page.

Examples of applications using AJAX: Gmail, Google Maps, Youtube, and Facebook tabs.

The Ajax calls are used to in order to display filtered information in the admin viewer, by querying specific details about logs. This way, we do not load the whole database and filter the information on the client, but use Ajax specific query each time new filtered information are needed. Thus, we save time and no-needed data transmission by loading only the needed data and only when it is needed.

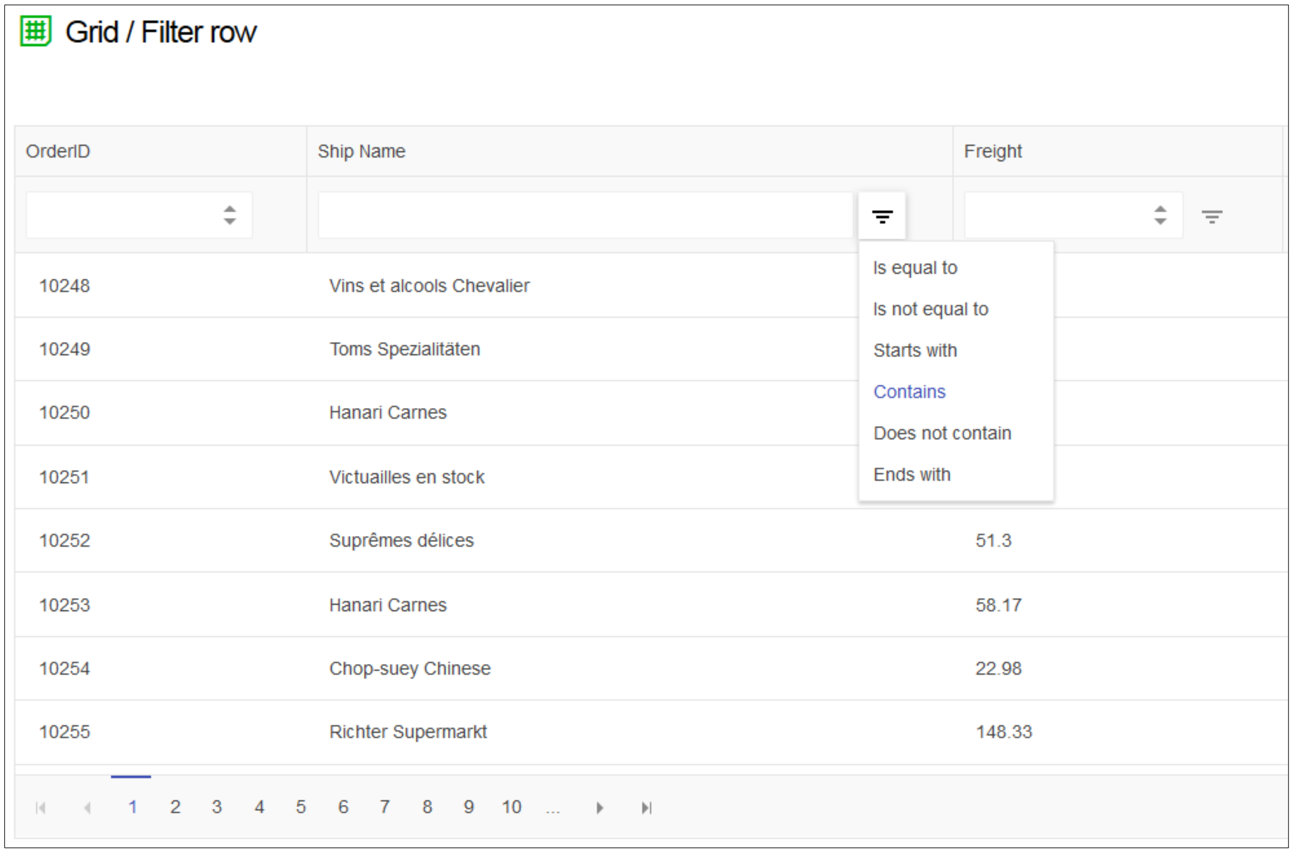
2.4.3 KendoUI

If the transmission of data between server and client is done using SignalR and Ajax, another technology is needed for us to display nicely and interactively the logs.

Kendo UI is a framework for modern HTML UI. Engineered with the latest HTML5, CSS3, and JavaScript standards, it delivers everything needed for client-side.

Kendo UI offers a large set of features for interactive UI like Charts generator, Drawer, Date Picker etc. but what is most suitable in our case is the Kendo UI Grid.

The grid offers everything is needed for the data to be displayed in an easy-to-understand and interactive way. Sort, Filter (server side or client side), Pagination, Export to Excel document, Read data from remote source etc. – all these are very useful features that are already implemented by Kendo and requires just a few lines of code to make use of them. Thus, by using Kendo, we can focus on the actual work on the log server and just pass the data to the grid, which will take care of it for us to display it nicely. It is an out-of-box, easy to implement and very elegant solution.



OrderID	Ship Name	Freight
10248	Vins et alcools Chevalier	
10249	Toms Spezialitäten	
10250	Hanari Carnes	
10251	Victuailles en stock	
10252	Suprêmes délices	51.3
10253	Hanari Carnes	58.17
10254	Chop-suey Chinese	22.98
10255	Richter Supermarkt	148.33

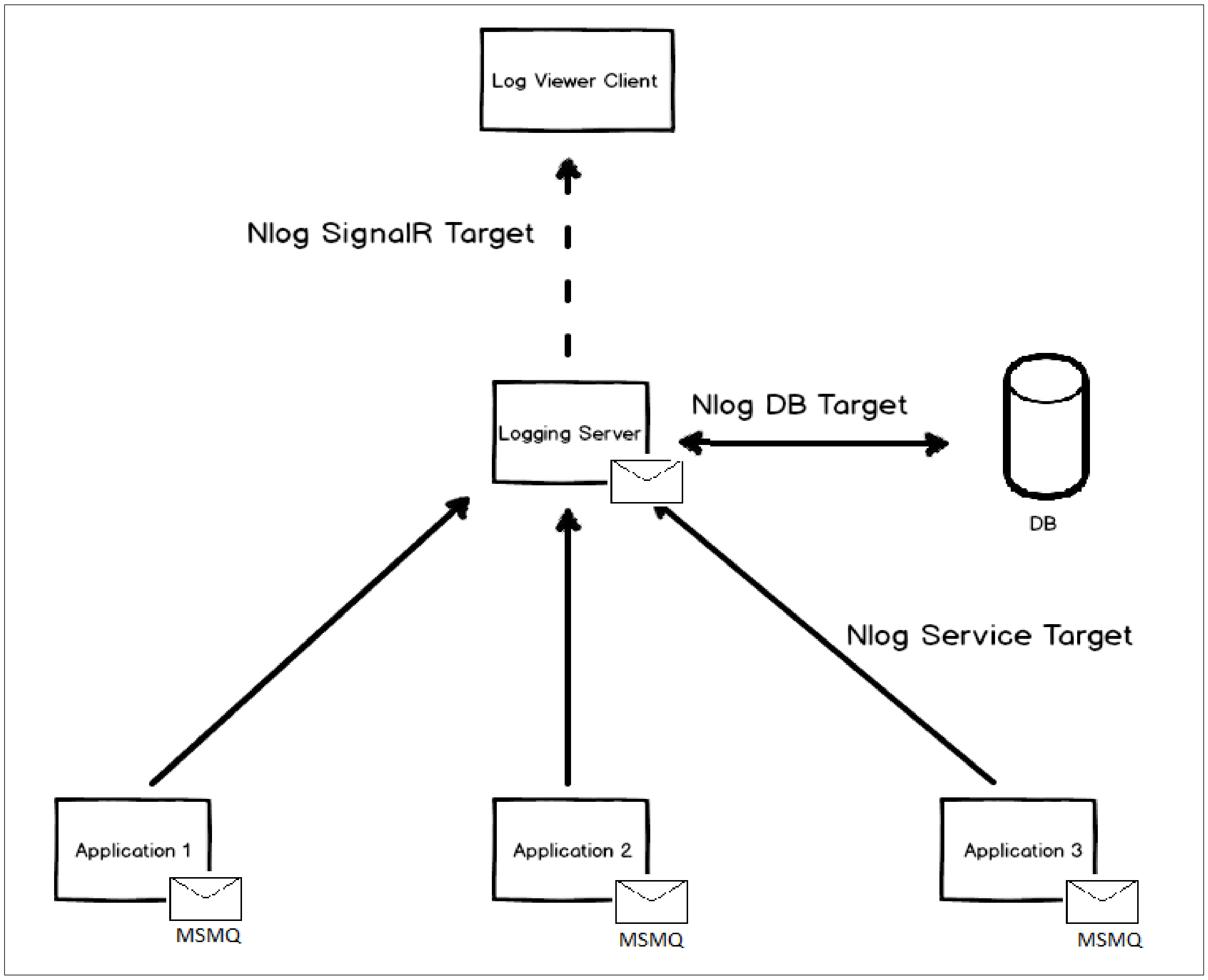
3. Analysis and Design

3.1 Server Architecture

Overview

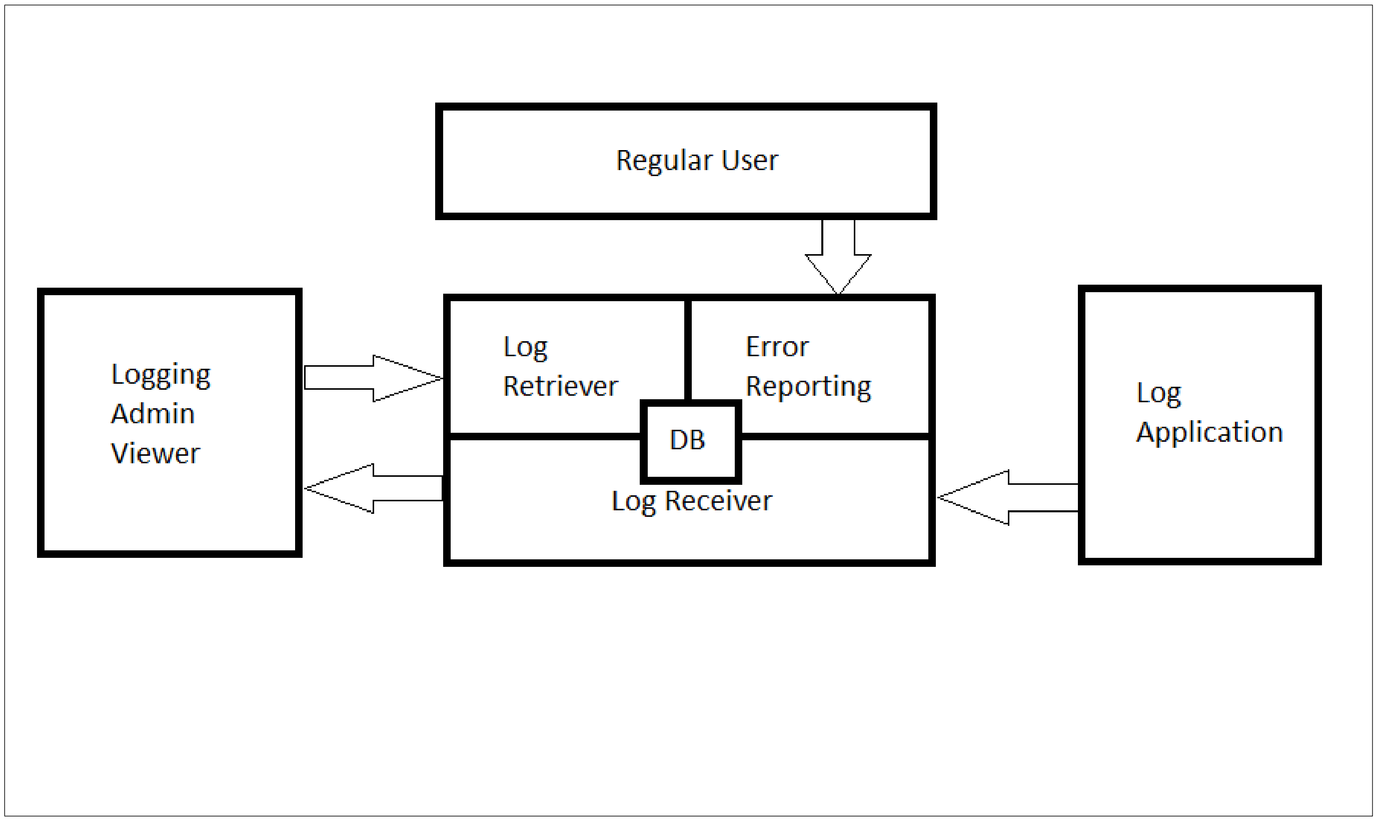
As it is specified earlier in the report, the server should be able to collect log messages from all applications that are connected to it; the server should be able to push this information further (both to a storage location which is the MSSQL Database and to the logging admin viewer). The server should also take care of the requests that it receives from the log admin viewer and return the logs stored in the Database. It should make sure that no sensitive information is sent to an inadequate application.

All these being said, we can draw a schema of how the whole server looks like and how it interacts with the connected applications that are logging to it.



Once we have a perspective image of how the logging server is interacting with the applications, database and the log viewer client, we can start looking at the server itself in more details:

- **Log Receiver Service** - Is the service used as target by the applications to log the information. The log receiver service is the one that processes the logs, and sends them further to the Log Viewer client using SignalR and to the database using NLog Database target. This service is dealing with the real-time logs.
- **Log Retriever Service** - Is the service used to retrieve information regarding logs from database. The service is used by the Logging Admin Viewer client to access the requested data. The log retriever service has to process the received query and return the logs to the Logging Admin Viewer. The service uses ADO.NET entity framework and WCF Data Service to query the DB. This service is dealing with the permanent logs that are stored in DB.
- **Error Reporting Service** – Is the service that returns the logs for the normal users in case of any error. The users will receive an URL to the download location, and will send the document further to the developer. The Error Reporting Service has to take care that no sensitive information are read by a regular user. This service is used by regular users to retrieve a snapshot of their error.



3.2 Logging Applications

3.2.1 Configuration

The configuration of the applications in order to send their log messages to the logging server has to be straight-forward, and preferable following a pattern that can be used for all of them.

The configuration should not change anything in the code and in the logic of the application.

The NLog framework can be configured both, by using code or XML file. There are no differences between them. This project is using the XML configuration for the ease of read and its tree structure.

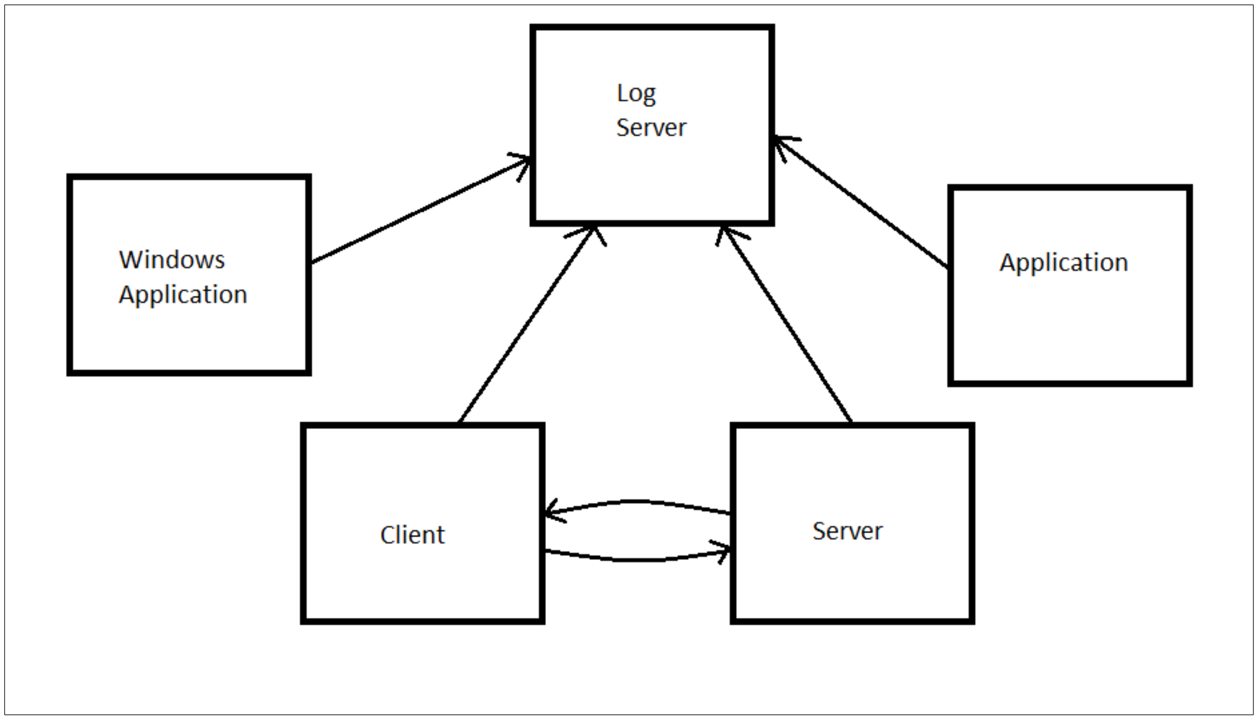
3.2.2 Client-Server application

In order to follow the logs in a client-server application it is important that both the server and the client are able to log.

3.2.3 Windows application

Not only web applications or servers but also the windows applications should be able to log.

It is important to understand that the Logging System can collect the logs from many applications, either if those applications are connected or not one to another.



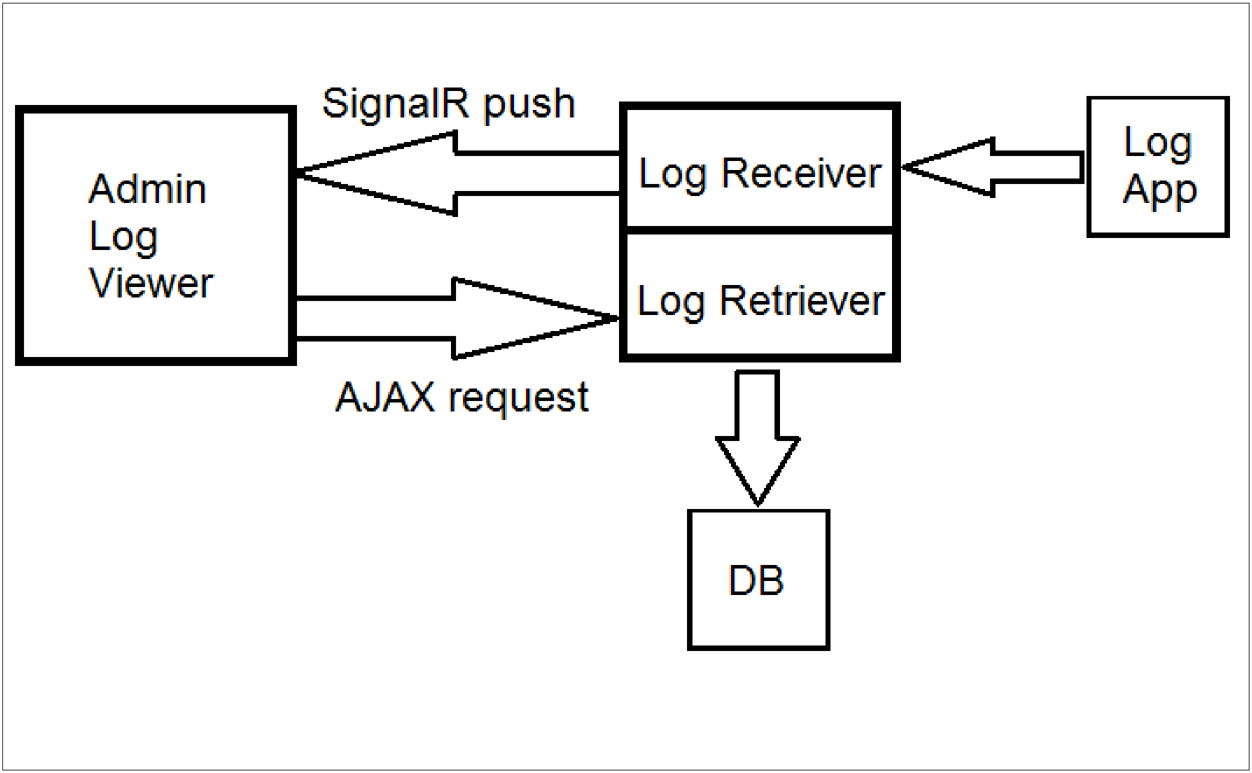
3.3 Admin Viewer

Data transmission and data display

As it is mentioned previously in the report, the transmission of logs between logging server and the logging admin viewer will be ensured by SignalR and Ajax calls.

Log Receiver Service, the one that collects the logs from the applications, will be configured this way that the NLog framework will send the logs to the dabatase and to a SignalR function. Once the SignalR function receives the logs, it will push them further to the Logging Admin Viewer. The Logging Admin Viewer, using the Kendo UI, will update the grid in real-time.

For the data that is already in the database and only needs to be filtered and displayed, KendUI Grid has the option to use a Web API as external datasource. This way, the query URL for the Log Retriever Service API is generated automatically by the Kendo Grid and the grid is automatically updated with the result. The only configuration that has to be done in the Kendo grid is the base URL of the Log Retriever Service.



4. Implementation

4.1 NLog Framework configuration

4.1.1 Installation

The NLog installation of framework is straight-forward by using NuGet Packages. In order to install NLog Right click on the project, **Manage NuGet Packages**, install **NLog**. This will install the NLog dll.

Optionally can be installed the **Nlog Configuration** package which is the configuration file of the framework.

4.1.2 NLog Config file

Nlog for Extended Profile is also optionally in most of the cases but it is needed in this project in order to use MSMQ target, which is not a target implemented by default by the NLog framework.

The basic structure of the NLOg.config file includes the header, the targets to which are the logs sent and the rules that the logger should follow:

```
<?xml version="1.0" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <targets>
        <target name="NAME" xsi:type="TYPE" <OTHER_SPECIFIC_CONFIGS></target>
      </targets>

      <rules>
            <logger name="*" minlevel="LEVEL" writeTo="TARGET_NAME" />
      </rules>
</nlog>
```

4.1.3 Targets

Targets are used to display, store, or pass log messages to another destination. There are two kinds of target; those that receive and handle the messages, and those that buffer or route the messages to another target. The second group are called 'wrapper' targets.
Most used targets in the Logging Server and the applications that are logging to it are:

1. LogReceiverService
2. MSMQ
3. Method
4. Fallback
5. Database

4.1.4 Logging an event

In order to log an event, an object of type Logger has to be created.

```
Logger _logger = LogManager.GetCurrentClassLogger();
```

The object is then used to log by accessing its methods.

```
_logger.Info("Pressing a button on the frontend..");
_logger.Trace("Exception on the backend");
```

Optionally, an object of type Exception can be used as a second parameter.

```
_logger.Error("Division error", e);
```

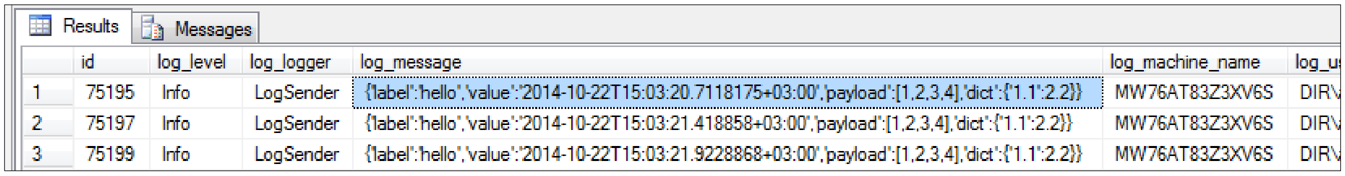
To attach a particular object as parameter to the log event, a method in this scope can be created in the LoggerExtensions class

```
public static class LoggerExtensions
{
        public static void Raw(this Logger logger, NLog.LogLevel loglevel, object x)
        {
                string json = JsonConvert.SerializeObject(x);
                json = json.Replace("\"", "'");
                LogEventInfo theEvent = new LogEventInfo(loglevel, logger.Name, json);
                theEvent.Properties["json"] = true;
                logger.Log(typeof(LoggerExtensions), theEvent);
        }
}
```

And then log the event:

```
_logger.Raw(NLog.LogLevel.Info, ComplexObject);
```

The attached object will be written in JSON format in the message field of the log object.



	id	log_level	log_logger	log_message	log_machine_name	log_u
1	75195	Info	LogSender	{'label':'hello','value':'2014-10-22T15:03:20.7118175+03:00','payload':[1,2,3,4],'dict':{'1.1':2.2}}	MW76AT83Z3XV6S	DIR\
2	75197	Info	LogSender	{'label':'hello','value':'2014-10-22T15:03:21.418858+03:00','payload':[1,2,3,4],'dict':{'1.1':2.2}}	MW76AT83Z3XV6S	DIR\
3	75199	Info	LogSender	{'label':'hello','value':'2014-10-22T15:03:21.9228868+03:00','payload':[1,2,3,4],'dict':{'1.1':2.2}}	MW76AT83Z3XV6S	DIR\

4.2 Server Implementation

4.2.1 Log Receiver Service

Is a WCF service used as target by the applications to log the information. The log receiver service is the one that processes the logs, and sends them further to the Log Viewer client using SignalR and to the database using NLog Database target.

Additionally to the NLog NuGet Package, the SignalR package has to be installed using NuGet.

Log Receiver NLog.config – has 2 main targets:

- FallbackGroup - contains the targets that will store the logs in DB
- HubMethod - target that sends the logs to the Logging Admin Viewer using SignalR

FallbackGroup - target is a wrapper for other targets. In order for the logs to not be lost in case of a target failure, inside the FallbackGroup can be specified many targets as alternatives and the information will be logged to the first available one.

```
<target xsi:type="FallbackGroup" name="DBFallback" returnToFirstOnSuccess="true">
       <target xsi:type=”Database” name=”DBtarget” OTHER_CONFIGURATION />
       <target xsi:type=”Database2” name=”DBtarget2” OTHER_CONFIGURATION />
       <target xsi:type=”File” name=”FileTarget” OTHER_CONFIGURATION />
</target>
<rules>
       <logger name="*" minlevel="LEVEL" writeTo=" DBFallback "/>
</rules>
```

HubMethod is a target of type `MethodCall`. The log server triggers the specified method whenever a log is made. In this case the method is part of a SignalR Hub which sends further, to the LogViewer, all the logs received by the Receiver Service. NLog can also use directly SignalR as target, but in order to do this **NLog.SignalR.dll** is required.

Example of the MethodCall target.

In the nlog.config file, a new target is created. The type has to be set to MethodCall, while the class name is the path to the class where the method that will be triggered is located. Additionally if the method has any parameters, those can be declared in nlog.config using the <parameter> tag. The paramters name does not matter, but only their order.

Nlog.config

```
<target name="hubMethod" xsi:type="MethodCall" className="LoggingReceiver.Hubs.LogHub,
LoggingReceiver, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
methodName="Send">
<parameter name="PARAMETER_NAME " layout="${VALUE}"/>
</target>
```

In C# code, the method that is being used as target by the nlog.config has to be declared and follow the same structure.

C#

```
public static void Send(string PARAMETER_NAME)
{
//code
}
```

Even if the NLog documentation may not specify it, sometimes the **Version, **Culture** and **PublicKeyToken** have to be added in order for the method to be triggered.*

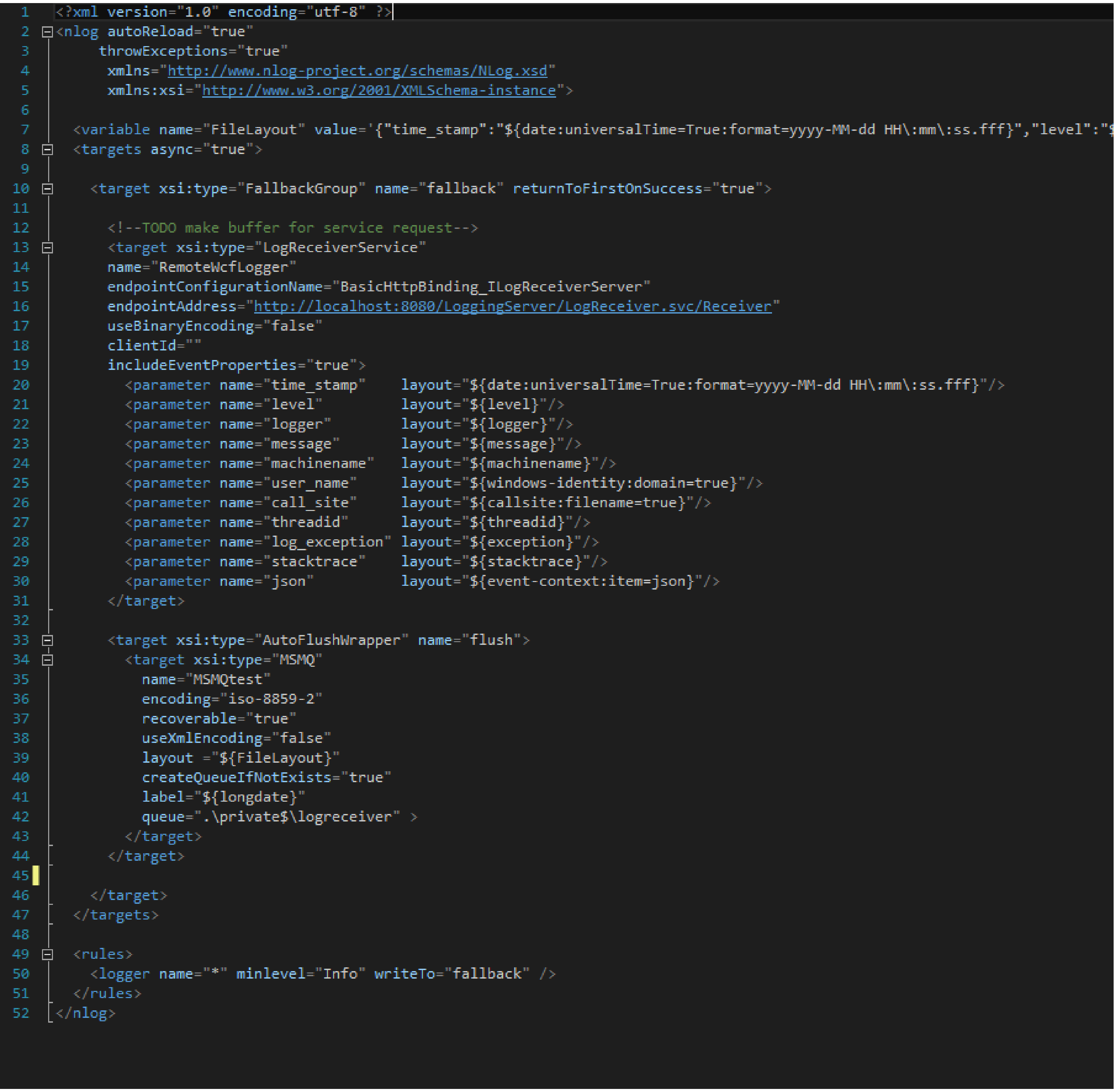
For the Log Receiver Service to be able to receive logs from other applications, through the `LogReceiverService` target, it has to implement a method that will process the NLogEvents and the class has to use the `ILogReceiverServer` interface.

```
public class ReceiverService : ILogReceiverServer
        public void ProcessLogMessages(NLogEvents nevents)
                {
                var events = nevents.ToEventInfo();
                foreach (var eachEvent in events)
                {
                        var logger = LogManager.GetLogger(eachEvent.LoggerName);
                        logger.Log(eachEvent);
                }
}
```

The ProcessLogMessages method is the one called by the NLog framework from the applications that logs. This method sends further the logs to the Logging Server NLog framework which processes them accordingly to the configuration file.

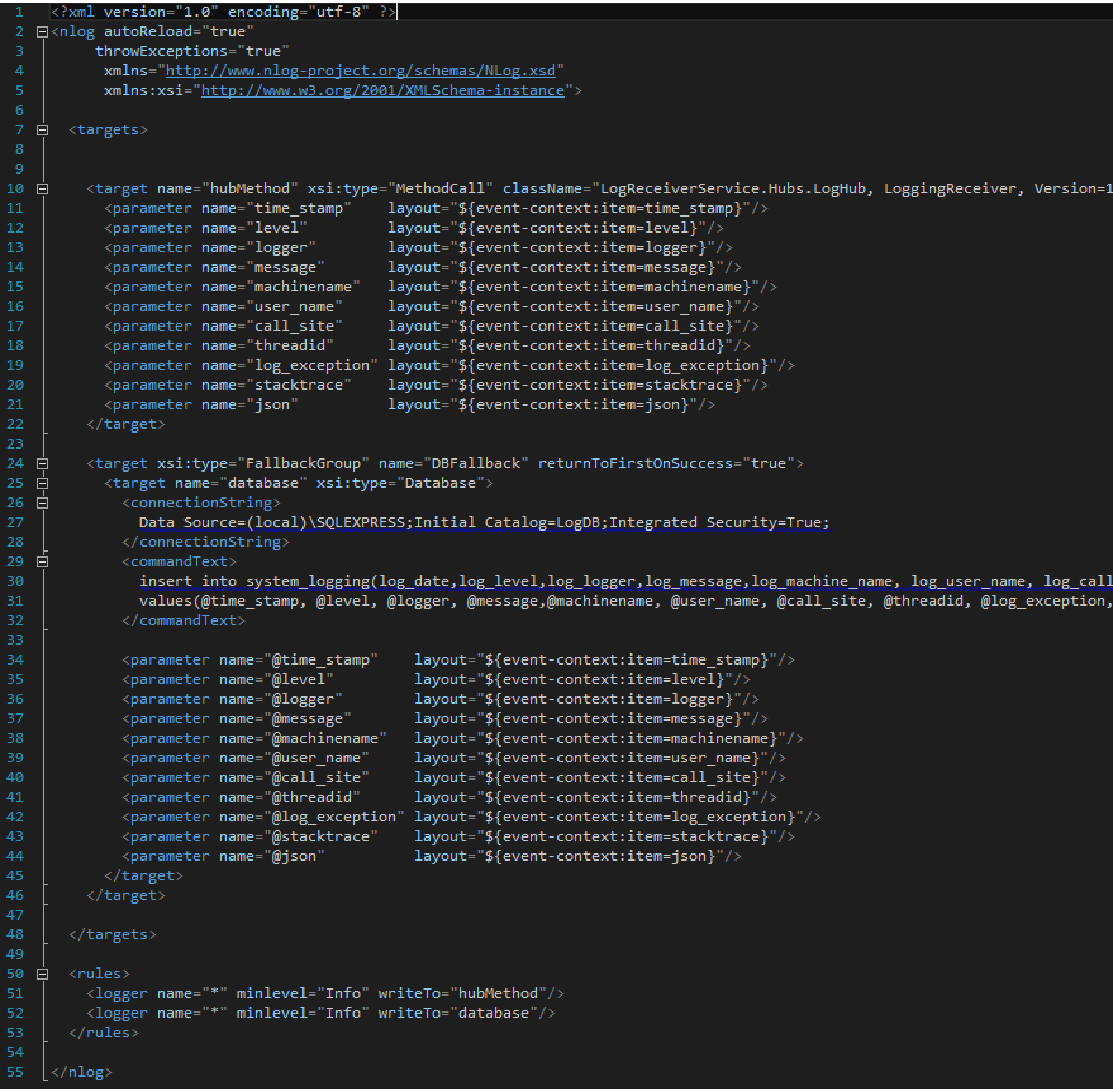
Further can be seen how the applications are configured so that their logs are sent to the Log Receiver Service, and how the Log Receiver Service is configured to process these logs (sent further to the Logging Admin Viewer and to the Database).

NLog configuration file of log application



```
<?xml version="1.0" encoding="utf-8" ?>
<nlog autoReload="true"
     throwExceptions="true"
      xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <variable name="FileLayout" value='{"time_stamp":"${date:universalTime=True:format=yyyy-MM-dd HH\:mm\:ss.fff}","level":"$
  <targets async="true">

    <target xsi:type="FallbackGroup" name="fallback" returnToFirstOnSuccess="true">

      <!--TODO make buffer for service request-->
      <target xsi:type="LogReceiverService"
      name="RemoteWcfLogger"
      endpointConfigurationName="BasicHttpBinding_ILogReceiverServer"
      endpointAddress="http://localhost:8080/LoggingServer/LogReceiver.svc/Receiver"
      useBinaryEncoding="false"
      clientId=""
      includeEventProperties="true">
        <parameter name="time_stamp"    layout="${date:universalTime=True:format=yyyy-MM-dd HH\:mm\:ss.fff}"/>
        <parameter name="level"         layout="${level}"/>
        <parameter name="logger"        layout="${logger}"/>
        <parameter name="message"       layout="${message}"/>
        <parameter name="machinename"   layout="${machinename}"/>
        <parameter name="user_name"     layout="${windows-identity:domain=true}"/>
        <parameter name="call_site"     layout="${callsite:filename=true}"/>
        <parameter name="threadid"      layout="${threadid}"/>
        <parameter name="log_exception" layout="${exception}"/>
        <parameter name="stacktrace"    layout="${stacktrace}"/>
        <parameter name="json"          layout="${event-context:item=json}"/>
      </target>

      <target xsi:type="AutoFlushWrapper" name="flush">
        <target xsi:type="MSMQ"
          name="MSMQtest"
          encoding="iso-8859-2"
          recoverable="true"
          useXmlEncoding="false"
          layout ="${FileLayout}"
          createQueueIfNotExists="true"
          label="${longdate}"
          queue=".\private$\logreceiver" >
        </target>
      </target>

    </target>
  </targets>

  <rules>
    <logger name="*" minlevel="Info" writeTo="fallback" />
  </rules>
</nlog>
```

Log Receiver Service - NLog configuration file



```
<?xml version="1.0" encoding="utf-8" ?>
<nlog autoReload="true"
     throwExceptions="true"
      xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <targets>


    <target name="hubMethod" xsi:type="MethodCall" className="LogReceiverService.Hubs.LogHub, LoggingReceiver, Version=1
      <parameter name="time_stamp"    layout="${event-context:item=time_stamp}"/>
      <parameter name="level"         layout="${event-context:item=level}"/>
      <parameter name="logger"        layout="${event-context:item=logger}"/>
      <parameter name="message"       layout="${event-context:item=message}"/>
      <parameter name="machinename"   layout="${event-context:item=machinename}"/>
      <parameter name="user_name"     layout="${event-context:item=user_name}"/>
      <parameter name="call_site"     layout="${event-context:item=call_site}"/>
      <parameter name="threadid"      layout="${event-context:item=threadid}"/>
      <parameter name="log_exception" layout="${event-context:item=log_exception}"/>
      <parameter name="stacktrace"    layout="${event-context:item=stacktrace}"/>
      <parameter name="json"          layout="${event-context:item=json}"/>
    </target>

    <target xsi:type="FallbackGroup" name="DBFallback" returnToFirstOnSuccess="true">
      <target name="database" xsi:type="Database">
        <connectionString>
          Data Source=(local)\SQLEXPRESS;Initial Catalog=LogDB;Integrated Security=True;
        </connectionString>
        <commandText>
          insert into system_logging(log_date,log_level,log_logger,log_message,log_machine_name, log_user_name, log_call
          values(@time_stamp, @level, @logger, @message,@machinename, @user_name, @call_site, @threadid, @log_exception,
        </commandText>

        <parameter name="@time_stamp"    layout="${event-context:item=time_stamp}"/>
        <parameter name="@level"         layout="${event-context:item=level}"/>
        <parameter name="@logger"        layout="${event-context:item=logger}"/>
        <parameter name="@message"       layout="${event-context:item=message}"/>
        <parameter name="@machinename"   layout="${event-context:item=machinename}"/>
        <parameter name="@user_name"     layout="${event-context:item=user_name}"/>
        <parameter name="@call_site"     layout="${event-context:item=call_site}"/>
        <parameter name="@threadid"      layout="${event-context:item=threadid}"/>
        <parameter name="@log_exception" layout="${event-context:item=log_exception}"/>
        <parameter name="@stacktrace"    layout="${event-context:item=stacktrace}"/>
        <parameter name="@json"          layout="${event-context:item=json}"/>
      </target>
    </target>

  </targets>

  <rules>
    <logger name="*" minlevel="Info" writeTo="hubMethod"/>
    <logger name="*" minlevel="Info" writeTo="database"/>
  </rules>

</nlog>
```

4.2.2 MSMQ fallback option

The Log Receiver Service has also a second role: to check whenever the **MSMQ** is not empty (which means that there was a problem with the log transaction using the ServiceTarget, between the applications and the server).

To reduce the complexity of the queue check, an event handler method can be attached to the queue so that every time its status is changing (new entry, change value, etc) that method will be triggered.

Whenever the server is started (or restarted), the `insertOfflineLogs` method is called from the **App_Code/Initializer.cs** class and subscribes to the queue for changes.

```
public static void insertOfflineLogs()
{
        string myQueue = ".\\private$\\LogReceiver";
        MessageQueue Q1 = new MessageQueue(myQueue);
        Q1.ReceiveCompleted += new ReceiveCompletedEventHandler(MyReceiveCompleted);
        Q1.BeginReceive();
}
```

**The logs are being written in the MSMQ in JSON format. The format is being declared manually using an NLog variable.*

If the queue is not empty, or logs are being written to it, the server will read the messages and insert them in to the DB. `public void Insert(nlogobj obj)`

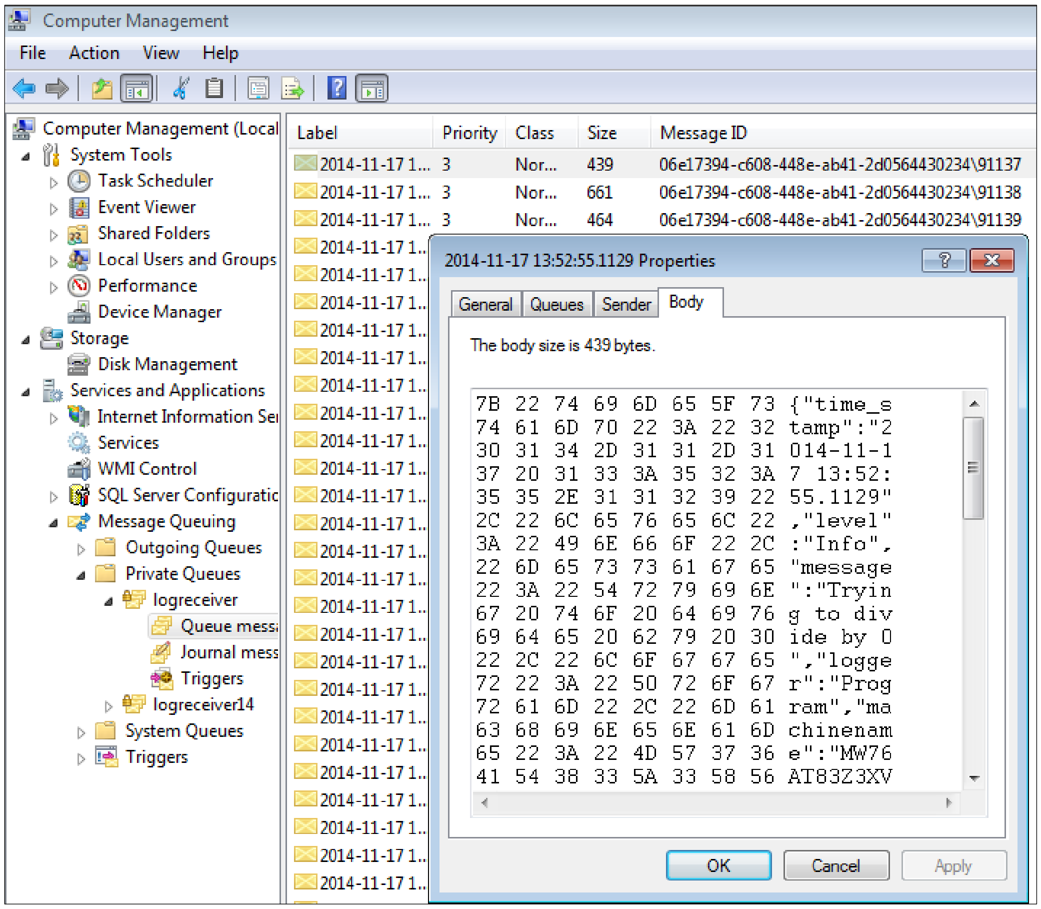
The **nlogobj** class is used to deserialize the MSMQ messages to an object that maps the DB. The deserialiazation can also be done also to a dynamic object, but this does not ensure that the MSMQ messages will map the DB and that no DB Query exceptions may occur.

Every application that is logging to the Log Server has its own local MSMQ to which is writing the messages if there is no connection with the server. When the connection is back, MSMQ takes care that all the local messages from queue are send to the destination (MSMQ queue of Log Server).

When the MSMQ queue of the Log Server is changing its state, the previous mentioned method is called, and all the messages are read, transformed from bytes to string, deserialiazed to nlog objects and inserted in to the database.

The messages in MSMQ can be saved in binary format or XML. The binary option was more suitable in this case because of the verbose XML metadata and the fact that MSMQ does not create individual tags for each element that the NLog framework is sending, but only one tag for all the details, thus the deserialiazation would have been harder. By formatting the logs to JSON manually, in code, and save it as binary to MSMQ we can easy deserialiaze and it takes less space.

Here can be seen how the logs have been saved in the local MSMQ because of the connection problem with the Log Server.



Label	Priority	Class	Size	Message ID
2014-11-17 1...	3	Nor...	439	06e17394-c608-448e-ab41-2d0564430234\91137
2014-11-17 1...	3	Nor...	661	06e17394-c608-448e-ab41-2d0564430234\91138
2014-11-17 1...	3	Nor...	464	06e17394-c608-448e-ab41-2d0564430234\91139
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				
2014-11-17 1...				

The body size is 439 bytes.

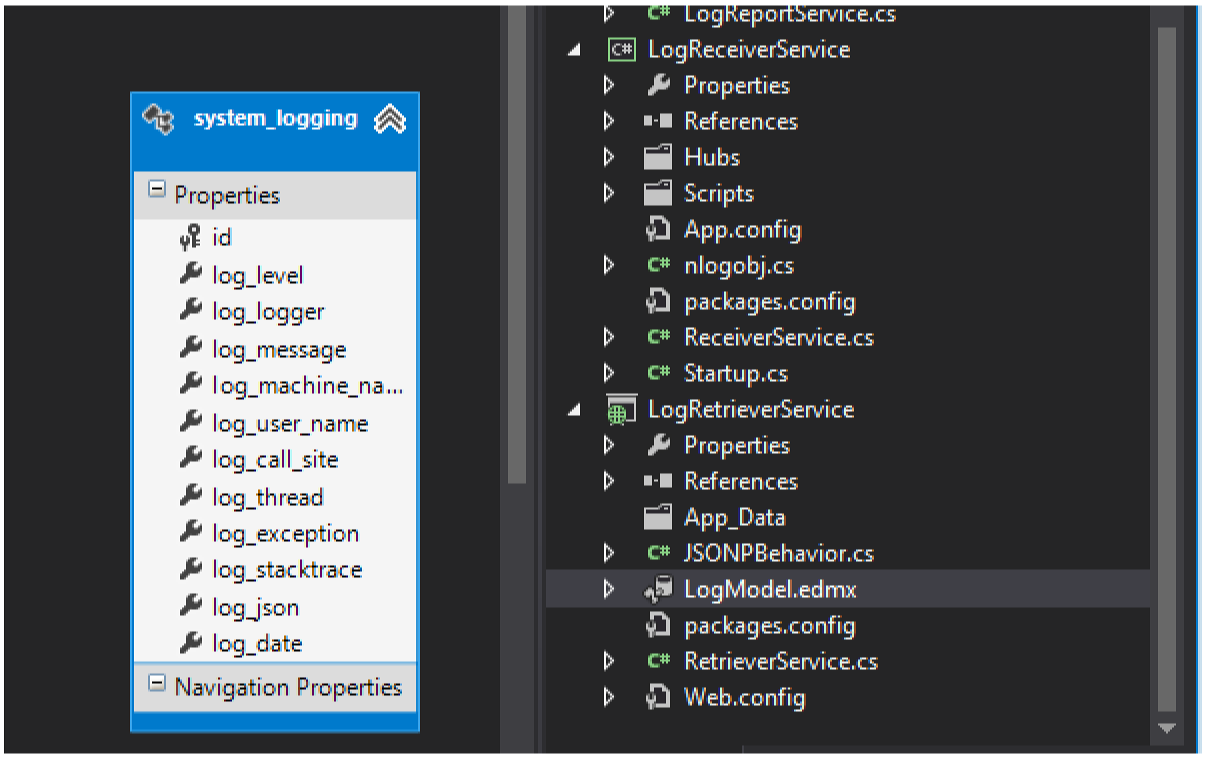
```
7B 22 74 69 6D 65 5F 73 {"time_s
74 61 6D 70 22 3A 22 32 tamp":"2
30 31 34 2D 31 31 2D 31 014-11-1
37 20 31 33 3A 35 32 3A 7 13:52:
35 35 2E 31 31 32 39 22 55.1129"
2C 22 6C 65 76 65 6C 22 ,"level"
3A 22 49 6E 66 6F 22 2C :"Info",
22 6D 65 73 73 61 67 65 "message
22 3A 22 54 72 79 69 6E ":"Tryin
67 20 74 6F 20 64 69 76 g to div
69 64 65 20 62 79 20 30 ide by 0
22 2C 22 6C 6F 67 67 65 ","logge
72 22 3A 22 50 72 6F 67 r":"Prog
72 61 6D 22 2C 22 6D 61 ram","ma
63 68 69 6E 65 6E 61 6D chinenam
65 22 3A 22 4D 57 37 36 e":"MW76
41 54 38 33 5A 33 58 56 AT83Z3XV
```

4.3 Log Retriever Service

Is the service used to retrieve information regarding logs from database. It is used by the Log Viewer client to display the requested data.

The Log Retriever is using ADO.NET Entity Data Model to map the database and WCF Data Service.

ADO.NET was the right choice to map the database because is easy to configure, the generated code has the guarantee of no-bugs, best-practices, best-performance and doing the mapping manually is a loss of time and effort which can used in more useful ways.



The WCF Data Service is the perfect tool in our situation, taking in consideration the KendoUI Grid which is going to display its results. The KendoUI Grid has the compatibility with OData Services, the only configuration that has to be done being the base URL of the service.

```
   [JSONPSupportBehavior]
    public class RetrieverService : DataService<EntitiesLog>
    {
        public static void InitializeService(DataServiceConfiguration config)
        {
         config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
         config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
        }
    }
```

One problem that occurred was that the OData service was returning by default the result in XML format, while the KendoUI was requesting JSON, thus the service had to be configured to return JSON by adding a JSONPBehavior class.

For the Log Viewer client to be able to display the data in the KendoUI grid the **$format=json** parameter has to be validated in the Data Service Web URI. To do this, the JSONPBehaviour class has to be added in the Log Retriever Project. The [`JSONPSupportBehavior`] has to be added in the Log Retriever Service class. The `DataServiceProtocolVersion` has to be set to V2.

4.4 Error Reporting Service

Is the service that returns the logs for the normal users in case of any error. The users will receive an URL to the download location, and will send the document further to the developer. The Error Reporting Service has to take care of the security issues that may occur.

The service is a Web API which exposes one method – getLogs and returns the logs as a stream of bytes that the user will save as a file. The ***ticks*** parameter is the date in .NET ticks format.

```
    [ServiceContract]
    public interface ILogReportService
    {
        [OperationContract]
        [WebGet(UriTemplate="/GetLogs={ticks}")]
        Stream GetLogs(string ticks);
    }
```

Since the logs may contain sensitive information which cannot be read by a regular user, the logs are encrypted using salt and a key. When the developer receives the logs from the user, he has to decrypt the logs in order to be able to understand them. This does ensure the security needs not only for the regular user access, but also for the transfer between the server and client since the logs are encrypted on the server, before being sent.

```
public Stream GetLogs(string ticks)
        {
            DateTimeOffset date = new DateTime(long.Parse(ticks));
            String datetime = date.ToString("yyyy-MM-dd HH:mm:ss.fff");
            var resultList = getLogs(datetime);
            string resultEncrypted = Encrypt(resultList, key);
```

For the response of the Error Reporting Service to be a download file, the response headers have to be edited

```
MemoryStream ms = new MemoryStream(System.Text.Encoding.ASCII.GetBytes(resultEncrypted));
WebOperationContext.Current.OutgoingResponse.Headers.Add("Content-Disposition",
"attachment; filename=Logs" + datetime.Replace(" ", "T") + ".txt");
WebOperationContext.Current.OutgoingResponse.ContentType = "application/x-file-to-save;
charset=UTF-8";
```

And then we can return the result as a stream of bytes that will be returned as a download file by the browser thanks to the response headers

```
return ms;
```

4.5 Applications to log

In order for the application to be able to send the logs to the server, the NLog reference has to be added by installing the NLog package from Nuget.

In the code, an object of type NLog logger has to be created in the needed class, and then calls to its methods to log the event:

```
private static Logger _logger =
LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType.Name);
```

```
_logger.Warn("Waning! A problem occurred on the system.", new SystemException());
```

The logging configuration can be done either by using the NLog.config XML file or programmatically. As it is mentioned previously, the XML option was chosen because of its ease to read, tree structure and NLog intelisense.

NLog Extended.dll has to be added so that MSMQ can be used as a valid target.

Once the references have be added, the NLog.config file has to specify the targets. Since we want all our logs to be centralized in a single place which is our server, there is only one target in this case (A FallbackGroup target) which has as first option the Log Receiver Service and the MSMQ as fallback option. More details about Service Target can be found on the official documentation.

The MSMQ target is also wrapped in an **AutoFlushWrapper** target which ensures that no messages are being lost in transaction by flushing after each write. The queue address has to be the address of the MSMQ from the Logging Server (details). Eg.:

- FormatName:DIRECT=OS:servermachine\PRIVATE$\ServerQueue

or

- FormatName:Direct=TCP:ipserveraddress\private$\ServerQueue

The layout parameter of the MSMQ target is the actual message. Since there is only one column to store all the information about a log, by formatting the details in to a known format as JSON will ensure easy deserialization on the server side.

By using an NLog variable (`${FileLayout}`) it is possible to define a pattern for the MSMQ message so that it can be processed automatically when is read by the server (nlogobj from Logging Service):

```
<variable name="FileLayout" value='{"time_stamp":"${date:format=yyyy-MM-dd
HH\:mm\:ss.fff}","level":"${level}","message":"${message}","logger":"${logger}","machinen
ame":"${machinename}","username":"${windows-
identity:domain=true}","call_site":"${callsite:filename=true}","threadid":"${threadid}","
exception":"${exception}","stacktrace":"${stacktrace}","json":"${event-
context:item=json}"}' />
```

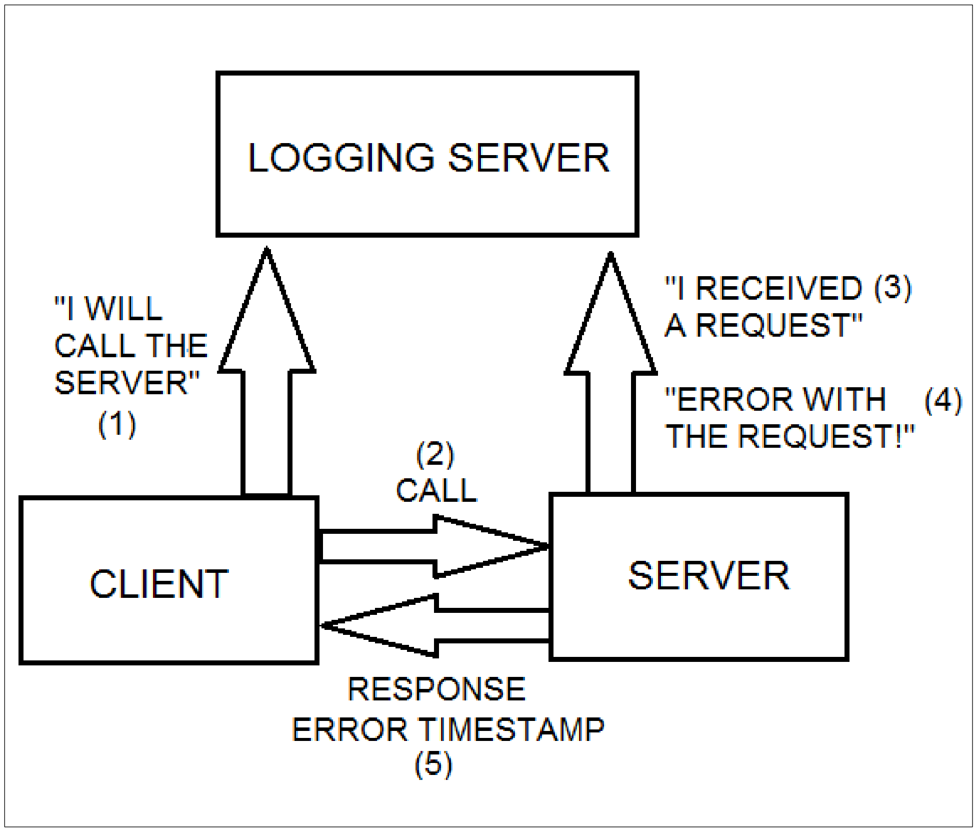
4.5.1 Allowing clients to download the logs

Most of the time the developer won’t have time to continuously check the Logging Admin Viewer, and even if it checks it he may miss some of the errors.

With all the filters and searching options that the Logging Admin Viewer offers, it might also take a while to search for a particular error.

Another reason might be that the customer may not allow the developer to have total access at the Logging Database/Logging Admin Viewer and the product owner does not want either that its employers to see the logs (security reasons). Thus, by implementing a system through which we are retrieving only the last hour/last 100 logs before the error occurred and ask the employees to send it to us, we can debug the application without having full access to the Log Database and without showing sensitive information to the employee.

This is done using the Error Reporting Service which returns all the logs from the last hour before the event occurred. There is need for more than just one log in order to understand the problem, since the context in which it happened is mandatory in the process of solve it – being able to understand what where the steps that lead to that particular unexpected event.


LOGGING SERVER
"I WILL CALL THE SERVER" (1)
"I RECEIVED A REQUEST" (3)
"ERROR WITH THE REQUEST!" (4)
(2) CALL
CLIENT
SERVER
RESPONSE ERROR TIMESTAMP (5)

4.5.2 Throw exception as log files to the front-end client

Throwing an exception that occurs in an application to the user will show him that there was a problem with his request and may suggest a solution for it.

However sometimes in distributed systems, the error may occur on another server and the user may not receive a response. In order to solve that, SignlaR can be used to push the error to the user while ToastrJs (**toastr** is a Javascript library for non-blocking notifications - jQuery is required) to display it.

Of course, instead of just throwing a generic error, we may as well use the NLog framework to return a well-documented message. But how good is this idea? Do we have enough space to display also the context of the error? Can we trust the user that he/she will use the information only to debug the problem and not in other scopes? One answer is sure: Never trust the user.

The ToastrJs library which displays interactive messages on the screen may be okay for short messages and alerts but not for long messages and definitely not suitable for a whole list of logs. So returning a file which contains all these seems a better idea.

Throwing the error as a log file will allow them to alert the developer about the problem occurred and send him as well the log file with all the information that he needs. By encrypting the logs with a key that we know will allow us to be sure that the user is not using the information to obtain any profit but only in the scope it was meant: to send it further to developer.

Once all these are clear, the idea is clear: once an error occurred on the backend, send the time stamp when it happened to the client application. The client application will display a pop-up error using ToastrJs that contains the download URL of the log file.

On the front-end

Create a SignalR function that will be called by the backend whenever an error occurs:

```
LogHub.client.alertMessage = function (message, logger)

{

        toastr.error(message);

}
```

On the back-end

In the code side, catch the errors and log them as we normal do:

```
try

{

        string result = (x / y).ToString();

}

catch (DivideByZeroException e)

{

        _logger.Error("Error on the hub",e);

}
```

In the NLog.config set a new target to point to a HubMethod that will call the client SignlaR function, and set its rules to be Error or greater. This way, whenever a log of level Error of greater is made, the Hub Method will be triggered.

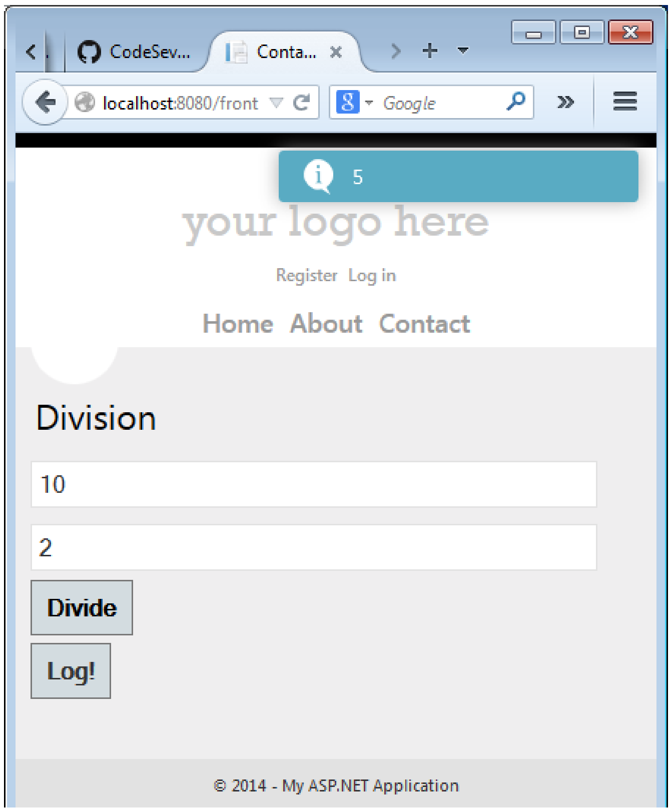
```
<targets>
        <target name="hubMethod" xsi:type="MethodCall" className="BackendWCF.Hubs.LogHub,
        BackendWCF, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
        methodName="send">
        <parameter name="message" layout="${message}"/>
        <parameter name="logger" layout="${logger}"/>
        </target>
</targets>

<rules>
        <logger name="*" minlevel="Error" writeTo="hubMethod" />
</rules>
```

Write the hub method mentioned in the NLog.config target

```
public static void send(string message, string logger)
{

        //Alert the clients that there is an error on the server
        GlobalHost.ConnectionManager.GetHubContext<LogHub>().Clients.All.alertMessage(mess
        age, logger);

}
```

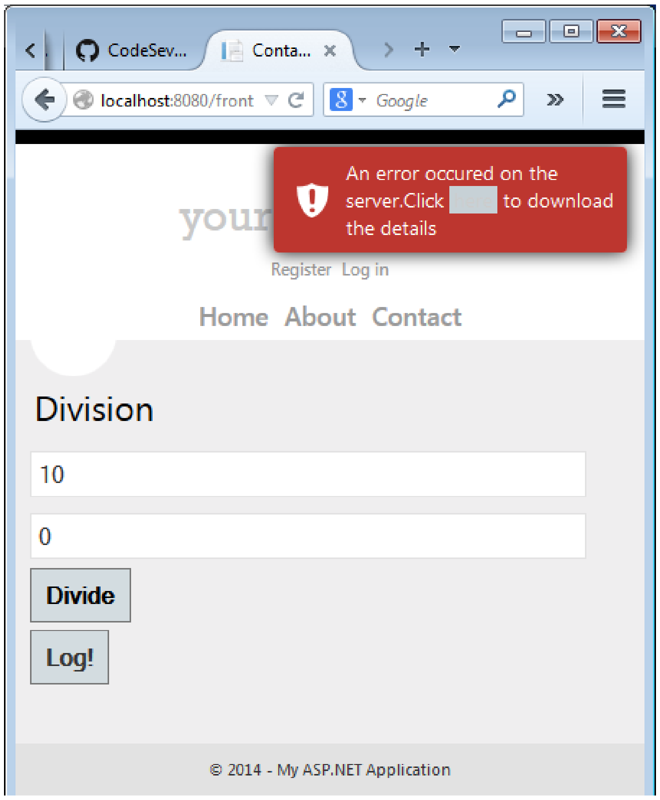
Now the Client Side SignalR function can use the returned result to display information regarding the error that occurred in the backend. Optionally, the Client Side may do an Ajax request to the Log Retriever Service and generate a file with the last hour logs so that the exception will have a context.

A simple example which uses a WCF service (Backend) and a ASP client (Frontend). The frontend makes a call to the Backend in order to divide 2 numbers. The backend returns the results which appears in a pop-up message using ToastrJS. In the first example the backend returns the result which is displayed in the pop-up message:



In the second example, the backend will throw an exception when we try to divide the number with 0.

(Divide by 0 exception).



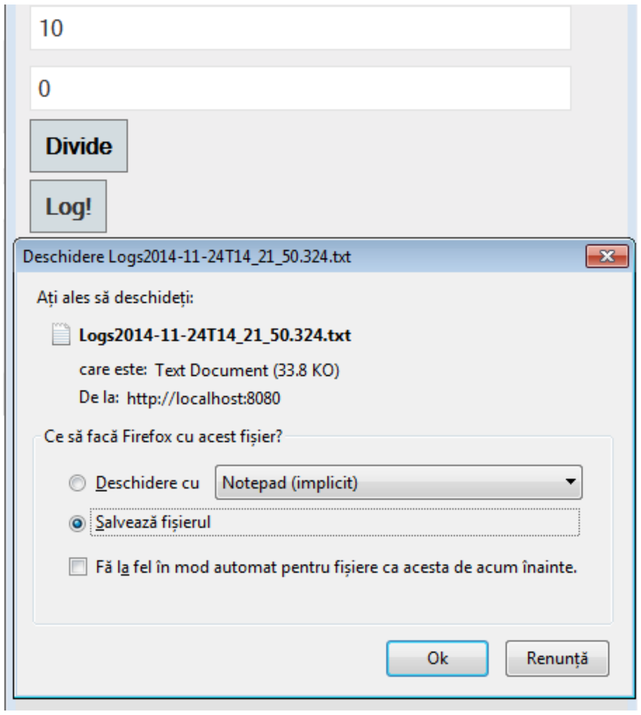
An error occured on the server.Click to download the details

Division

10

0

After we click the error message, we can download the file containing the log details



10

0

Deschidere Logs2014-11-24T14_21_50.324.txt

Ați ales să deschideți:

Logs2014-11-24T14_21_50.324.txt

care este: Text Document (33.8 KO)

De la: http://localhost:8080

Ce să facă Firefox cu acest fișier?

Deschidere cu Notepad (implicit)

[✓ Salvează fișierul]

Fă la fel în mod automat pentru fișiere ca acesta de acum înainte.

From the last 3 log messages in the files, we ca read our error

- 82979 Error LogHub Error on the hub MW76AT83Z3XV6S IIS APPPOOL\DefaultAppPool BackendWCF.Hubs.LogHub.divide(c:\Users\andrei.agape\Documents\Visual Studio 2012\Projects\BackendWCF\BackendWCF\Hubs\LogHub.cs:41) 54 Attempted to divide by zero. <>c__DisplayClass1.<WrapVoidAction>b__0 => <no type>.lambda_method => LogHub.divide False 11/24/2014 2:21:50 PM

- 82977 Info LogHub On the server side, preparing to divide MW76AT83Z3XV6S IIS APPPOOL\DefaultAppPool BackendWCF.Hubs.LogHub.divide(c:\Users\andrei.agape\Documents\Visual Studio 2012\Projects\BackendWCF\BackendWCF\Hubs\LogHub.cs:28) 54 <>c__DisplayClass1.<WrapVoidAction>b__0 => <no type>.lambda_method => LogHub.divide False 11/24/2014 2:21:50 PM

- 82978 Info LogHub Trying to divide 10 with 0 MW76AT83Z3XV6S IIS APPPOOL\DefaultAppPool BackendWCF.Hubs.LogHub.divide(c:\Users\andrei.agape\Documents\Visual Studio 2012\Projects\BackendWCF\BackendWCF\Hubs\LogHub.cs:32) 54 <>c__DisplayClass1.<WrapVoidAction>b__0 => <no type>.lambda_method => LogHub.divide False 11/24/2014 2:21:50 PM

Of course, those logs are what the developer sees only **after** he decrypts the file received from the employee. A normal user would see only unreadable encrypted text like this:

0qG0lHrR48eDpO+XM/u6kkHHyDfhGyFZJzkO6YFLaQPNH/TonWf8u048o8xKb2EVs98KVtG7gH0UA17VTzf
KEV4/T1k7Wns4dkx4rB9BI21wAewzipsPblr24mhU3VRUYFi4f9JKvh/7JM1s1eR8eb1bBBikTKAb1aiVg/9yxy
O2a4ZVBRGPvq7EIAa+iZczUS1wtzSEsT4eXOK6oKt6dfNktQ8a6QaCrW2RjqU/FmXGxnFpaAGLMek5kyfN6y
7V85j8cYwxyQ3RKIpExBeALWurOnZ4jIDPmbkOw3pe9SfKV52S33kRBe+6ylIch38fOp35y7ziJEJWLS22BDtS
4Lwnqm7UIFx+ielCpbysjfhryhtk561/7IEDkRANJ2smQbdhDJH15U9hCjZDMT+k/yahBj+bui0iuj1wJ6dQrl8A
kZ2kindjxcVkRz9cmOz3tLjk1eJ8mHJWXzHphi1OY+LPerGBeIt/WaO+fkO+q+6Q8y96AqT8U28zes/FniTjNsA
kDvGE+gsA9QbK3Lyt0jH+nB9pCCusdMkiOIanCjj9e3DupLFan2mpza3fK3CwyWyMO3WUffAhVpPKHsKFI0

4.6 Logging Admin Viewer

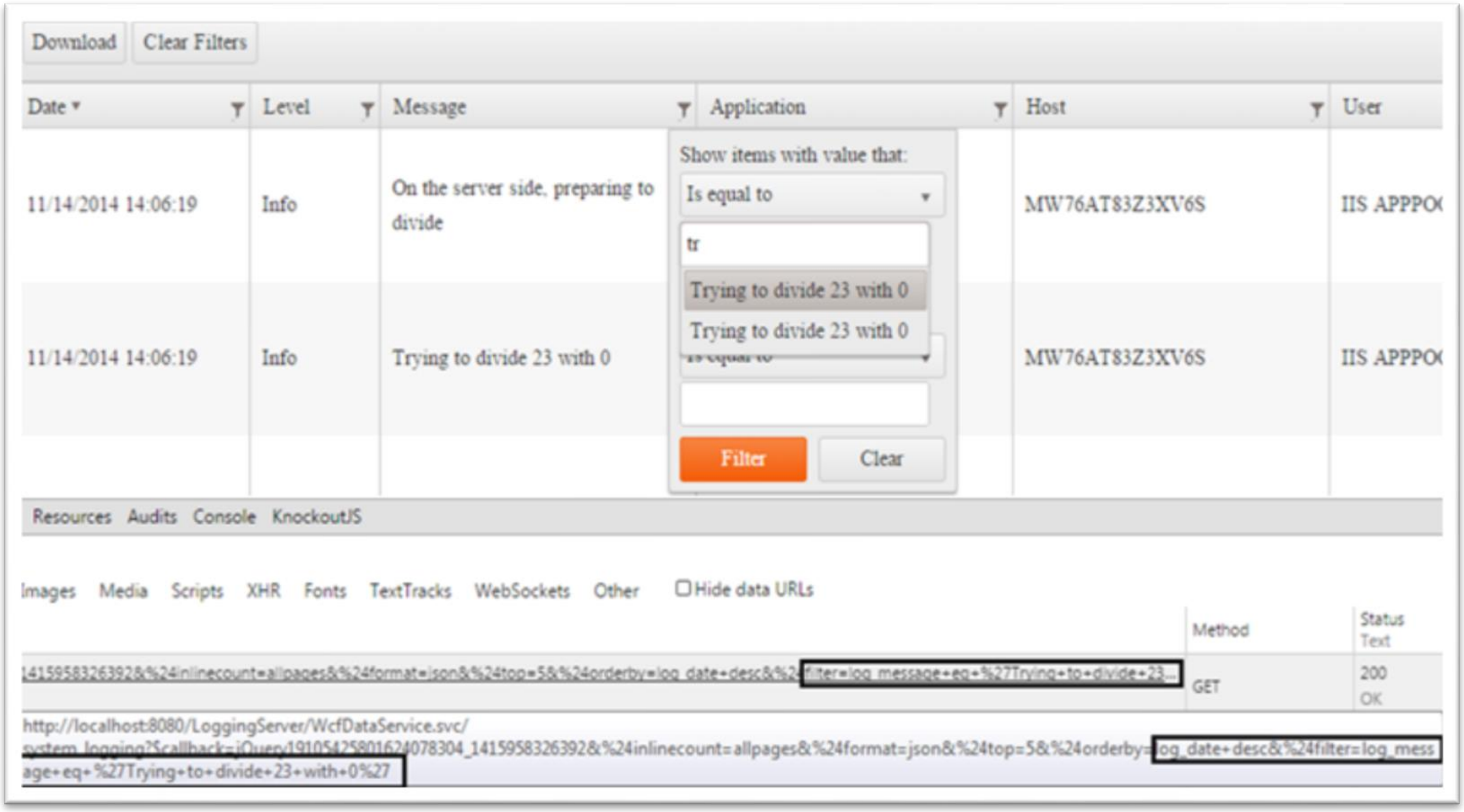
Is the application that displays real time information regarding any logs that have been made by the applications. The client is using KendoUI grid to display the logs and SignalR /Ajax calls to get the logs from the server.

The Kendo grid reads and displays the datasource using the address of the Log Retriever Service. By setting the datasource *type* to *odata*, the grid will automatically generate the query URI in order to request the logs that fit the filters criteria.

```
dataSource: { type: "odata",
```

Even if the grid shows that total number of logs that fit the searched criteria, the dataservice query is requesting only the number of logs that can be displayed on one page (*top=5* in this case). The filters are being applied on the server data, not on the client kendo datasource.

To download all the logs that fit the criteria, the dataservice URI call is saved in a variable, and another request is made, with the TOP parameter modified. In this way, the log file will have the most recent data and the service call will request a significant amount of data only when the logs are being downloaded. The .txt logs file is being generated on the client side using javascript.



Date	Level	Message	Application	Host	User
11/14/2014 14:06:19	Info	On the server side, preparing to divide		MW76AT83Z3XV6S	IIS APPPO(
11/14/2014 14:06:19	Info	Trying to divide 23 with 0		MW76AT83Z3XV6S	IIS APPPO(

	Method	Status Text
1415958326392&%24inlinecount=allpages&%24format=json&%24top=5&%24orderby=log_date+desc&%24filter=log_message+eq+%27Trying+to+divide+23...	GET	200 OK

```
http://localhost:8080/LoggingServer/WcfDataService.svc/
system_logging?$callback=jQuery19105425801624078304_1415958326392&%24inlinecount=allpages&%24format=json&%24top=5&%24orderby=log_date+desc&%24filter=log_message+eq+%27Trying+to+divide+23+with+0%27
```

4.6.1 Kendo Grid configuration

Kendo grid configuration can be done pretty straight forward by using JSON. The elements names are suggestive and easy to configure while the documentation is clear and helpful. To set the kendo grid to use the Log Retriever Service as *datasource* the base URL has to be specified and the data type has to be set as "***odata***" as it can be seen:

```
dataSource: {
       type: "odata",
       transport: {
       read:
       {
       url: "http://localhost:8080/LoggingServer/WcfDataService.svc/system_logging",
       complete: function (jqXHR, textStatus) {
              URL = this.url; //SAVES THE URL
       }
}
```

Once the datasource is set, optionally a data schema can be declared. The schema specifies what is the data type of each column. If a schema is not declared, the default is string, thus we couldn't have filter date or number column that is the reason why it was need to declare this explicitly:

```
schema: {
    model: {
        fields: {
             log_date: { type: "date" },
             log_level: { type: "string" },
             log_logger: { type: "string" },
             log_message: { type: "string" },
             log_machine_name: { type: "string" },
             log_user_name: { type: "string" },
             log_call_site: { type: "string" },
             log_thread: { type: "number" },
             log_exception: { type: "string" },
             log_stacktrace: { type: "string" }
         }
    }
},
```

Kendo Grid has the option to specify as well where the filtering should be done (server/client), enable the sorting and what is the default sort or the number of rows displayed ono ne page. Many other features are available, those being presented in order to get a general idea about the tool that has been used***:***

```
pageSize: 5,
serverPaging: true,
serverFiltering: true,
serverSorting: true,
sort: { field: "log_date", dir: "desc" }
```

4.6.2 Download the logs

As specified earlier in the report, sometimes the developer may need just a snapshot of the logs, download it and copy it to an external storage device. To do this, we will make use of the filtered data in the kendo grid and make another service call, this time for all the data the fit the criteria not only for the data the fits on one page.

After we receive the data, it is formatted using javascript and generated an in-memory file that can be downloaded. The log file contains the filters applied to the grid.

```
$("#download").click(function () {
var logsDownloadURL = URL.replace("top=" + numberOfRows, "top=" + logsToDownload);
        $.ajax({
            url: logsDownloadURL,
            type: "GET",
            dataType: "jsonp",
            async: false,
            cache: false,
            timeout: 30000,
            success: function (result) {
                var arr = result.d.results;
```

Formatting the text before generating the file:

```
                for (var i = 0; i < arr.length; i++) {
                    str += "[" + new Date(parseInt(arr[i].log_date.substr(6))) + "] ";
                    str += "[" + arr[i].log_level + "] ";
                    str += "[" + arr[i].log_message + "] ";
                    str += "\r\n";
                }
            },
            error: function () {
                console.log("error in generating logs for download file - ajax call")
            }
        });
```

Call the function that generates the file with “str” parameter containing the logs as a string

```
        createDownloadLink("#download", str, "dinesh.txt");
```

In this example, only the timestamp, the log level and the log messages are written to file. The formatting of text is optionally but makes the reading of files way easier.

Further is presented the function that generates the file, using a BLOB object.A Blob object represents a file-like object of immutable, raw data. Blobs represent data that isn't necessarily in a JavaScript-native format. The File interface is based on Blob, inheriting blob functionality and expanding it to support files on the user's system.

An easy way to construct a Blob is by invoking the Blob() constructor. Another way is to use the slice() method to create a blob that contains a subset of another blob's data.

4.6.3 JavaScript file generation

```
    function createDownloadLink(anchorSelector, str, fileName) {

        if (window.navigator.msSaveOrOpenBlob) {
            var fileData = [str];
            blobObject = new Blob(fileData);
            $(anchorSelector).click(function () {
                window.navigator.msSaveOrOpenBlob(blobObject, fileName);
            });
        } else {
            var url = "data:text/plain;charset=utf-8," + encodeURIComponent(str);
            $(anchorSelector).attr("href", url);
        }
    }
```

In order to update the Kendo grid with real time logs, the Log Receiver Service calls the SignalR function from the client and sends the corresponding data.

First of all, the NLog has to configured to send the logs to the SignalR Hub method.

Service NLog

```
<logger name="*" minlevel="Info" writeTo="hubMethod"/>
```

Then, in the SignalR method, we serialize the received log object to JSON and send it further to the Logging Admin Viewer

Service SignalR Hub

```
        public static void SendLog(string time_stamp, string level,…)
        {
            nlogobj obj = new nlogobj();
            obj.time_stamp = time_stamp;
            obj.level = level;
            //Same for all parameters
            string jsonNlogObj = JsonConvert.SerializeObject(obj);
GlobalHost.ConnectionManager.GetHubContext<LogHub>().Clients.All.displayLog(jsonNlogObj);
        }
```

Finally on the Logging Admin Viewer the function called by the hub has to be declared

Client SignalR function

```
    LogHub.client.displayLog = function (jsonNlogObj) {
        var model = createModelLog(jsonNlogObj);
        insertRow(model);
    };
```

5. Testing

The testing strategy was most based on stress tests, most common used being:

- Stopping and restarting the logging server while the applications where still log
- Multiple logs at the same time
- Big amount of logs in a short interval of time

Those tests were run both on a client-server application and on a windows application.

The client-server application sends logs from both the client and the server. Logs from a client are done both, before calling the server and after receives a response from it. The server logs when it receives a call from the client and after the operation is done.

While the logging server was stopped, both server-application and the client-application switched their NLog target to the MSMQ, thus no messages were lost. When restarted, the logging server received the offline messages and continued to get as well the logs that were made at that time. The logging server and the client-server application passed the test as everything went as expected.

For the second test, another client-server application was configured on another computer that was also pointing to the same Log Server. While both client-server applications were running at the same time, no log lost was observed.

For the third test, when a big amount of data was sending to the logging server on a short amount of time, a message was logged from inside a loop.

```
for (int i = 0; i < 3000; i++)
{
        _logger.Info(“Inside loop, iteration:” + i);
}
```

No loss of the log messages was observed.

The same tests were done on a windows application and the tests were passed successfully except the fallback writing to the MSMQ when in case of big amount of data, some messages were lost.

The problem was solved by adding a wrapper that flushes the logs.

```
<target xsi:type="AutoFlushWrapper" name="flush">
```

The testing has been done through the whole period of development and it was meant to show if the server does not present any new bug that may appear while a new feature was implemented and to see if the stress tests can be pass.

6. Problems encountered

6.1 NLog support

Even if the NLog documentation is well structured and contains information regarding many of the targets and arguments, not all these information are clear and may lead to misunderstandings.

Another problem encountered with the Nlog documentation was that most of the examples were to simple and less probably they may be enough for real life situations, while the discussions on support forums can be hardly found.

One of the problems encountered, and that was also mentioned earlier in the report, was the ***Method Call*** target of the Nlog configuration file. The official documentation does not mention that in some cases 3 additional parameters are needed (Version, Culture, PublicKeyToken) and neither how these values can be found.

```
<target name="hubMethod" xsi:type="MethodCall" className="LoggingReceiver.Hubs.LogHub,
LoggingReceiver, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
methodName="Send">
```

```
Insetead of
```

```
<target name="hubMethod" xsi:type="MethodCall" className="LoggingReceiver.Hubs.LogHub,
LoggingReceiver," methodName="Send">
```

6.2 NLog LogReceiverService target with MSMQ communication

An ideal solution of the NLog server would have include that the NLog framework can use the MSMQ as communication protocol and take care of the Queue creation, reading and deletion. Unfortunately, in order to use MSMQ for a service method, the Operation contract has to specify the **IsOneWay** parameter as **true**, but since the NLog service method that is taking care of the received logs is implementing a framework interface method that does not specify the **IsOneWay= true** parameter as true, the only solution to solve this problem was to manage the MSMQ log transmission manually.

MSMQ required method signature

```
[OperationContract(IsOneWay = true)]

Public void processLogs(string log)
```

Nlog framework interface

```
public class LogReceiverServer : ILogReceiverServer
   {
       public void ProcessLogMessages(NLogEvents nevents)
```

Since the **ILogReceiverServer** interface for **ProcessLogMessages** method does not **explicitly** specifies the IsOneWay true, the **default is false** and thus MSMQ transmission protocol cannot be used for the NLog method.

7. Conclusion

7.1 Implemented features

As predicted and anticipated at the beginning of the project, most of the features that were required have been successfully implemented. The server is now able to collect the logs from many distributed applications and is easy to implement without affecting the existing code.

The replacement of the existing way of logging can be easy done; once the server itself has been configured, the NLog NuGet package can be installed on the application and the configuration file can be copied from one to another.

The caching of the logs when the server is down or the connection cannot be made has been ensured by the MSMQ transmission protocol that makes sure that the log is not deleted until is not received by the recipient.

By implement a simple and intuitive GUI client, the logs can be accessed in a single place regarding the connected applications, can be filtered by many criteria and downloaded. The same client it is also display real time logs in order to anticipate and prevent system failures or security attacks.

7.2 Possible improvements

Since this is only a proof of concept, many other features and improvement can be implemented in a real distributed logging server as:

- **Download logs to Excel file** – downloading the logs in a excel file instead of a txt one can make the developer work easier while the information are more structured and the filter criteria can be used again without the need of returning to the Log Viewer Client
- **Send notifications of critical errors to developer email –** fourtanely the NLog framework makes available the email target, the only configuration that may need to be done, is adding a new target to the one existing, specifying the email address and the level error that will have to be exceed to send the emails (debug, critical, fatal etc.)
- **Display objects in JSON format inside the Logging Admin Viewer Grid in a more structured way instead of a string –** since sometimes the logging application is also sending an object together with its message, the display of that object may be a little tricky inside a Grid View. Displaying it as a JSON object may make it easier to read by the developer who analyze the problem

7.3 Working environment and feedback

Feedback regarding new features

Since I worked at this proof of concept in a company environment (Accenture Romania) I could easily see the differences between a university project and a real life one. The project idea has been recommended to me by the company and I find it interesting and challenging because it is not only developing my programming skills but also because it helps the firm to decide whether to implement a distributed logging server or to keep the current way of logging.

Ideas for architecture and improvements

Even before started, I had discussions with a few colleagues regarding the architecture of the whole system and the features and had to be implemented. I and my colleagues came with ideas that were discussed and we decided together which approaches are the best in order to develop this proof and concept because the idea may be used in the future for a real implementation if the concept is considered a success.

Support in using new technologies

While developing the server, I encountered many new technologies for me (MSMQ, SignalR, KendoUI) and I received great support from my colleagues not only in solving the bugs and problems but also to understand the concepts which I think it is even more important. This implementation of the logging server was a great theoretical and practical lesson for me and I am satisfied with both: the gained knowledge and the “final product”

8. References

Similar projects

- http://jasonwilder.com/blog/2012/01/03/centralized-logging/

Log4net or NLog

- http://stackoverflow.com/questions/710863/log4net-vs-nlog
- http://www.alteridem.net/2010/11/04/why-i-switched-from-log4net-to-nlog/
- http://essentialdiagnostics.codeplex.com/wikipage?title=Comparison

Fallback options: MSMQ

- http://msdn.microsoft.com/en-us/library/ms711472%28v=vs.85%29.aspx

Real-Time data transmission

- SignalR http://signalr.net/
- Ajax calls http://www.w3schools.com/jquery/jquery_ajax_intro.asp
- Kendo http://demos.telerik.com/kendo-ui/grid/filter-row

ToastrJS

- https://github.com/CodeSeven/toastr

BLOB

- https://developer.mozilla.org/en-US/docs/Web/API/Blob

GUID

- http://en.wikipedia.org/wiki/Globally_unique_identifier