Lappeenranta-Lahti University of Technology LUT

School of Engineering Science

Degree Programme in Software Engineering

Juuso Hautapaakkanen

IMPROVING INFORMATION RETRIEVAL IN A COMPANY INTRANET

Examiners:	Associate Professor Jouni Ikonen
	M.Sc. (Tech.) Ari Hämäläinen
Supervisors:	Associate Professor Jouni Ikonen
	B.Eng. Matti Jaatinen

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Degree Programme in Software Engineering
Juuso Hautapaakkanen

Improving information retrieval in a company intranet
Master's Thesis
2020

94 pages, 8 figures, 16 tables, 9 listings, 2 appendices

Examiners : Associate Professor Jouni Ikonen
M.Sc. (Tech.) Ari Hämäläinen

Keywords: information retrieval, search engine, intranet

Companies accumulate large amounts of information in their day-to-day operations and store it in various formats. With the rising amount of information, the efficiency of its retrieval should keep up. Often, though, the organization of the information becomes increasingly difficult, and retrieving it more and more troublesome. At such time it becomes necessary to look for ways to improve the retrieval of information at the least. This master's thesis identifies the development needs of a Finnish company in regard to information retrieval, investigates alternative solutions for fulfilling those needs and implements the deployment of one of these alternatives. The deployed intranet search engine improves the company's information retrieval from the server's network drives. The identified development needs reveal shortcomings in the management of knowledge, information and documents as well, the development of which the company is recommended to invest in in the future.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Ohjelmistotuotannon koulutusohjelma
Juuso Hautapaakkanen

Tiedonhaun parantaminen yrityksen sisäverkossa
Diplomityö
2020

94 sivua, 8 kuvaa, 16 taulukkoa, 9 katkelmaa, 2 liitettä

Työn tarkastajat:	Tutkijaopettaja Jouni Ikonen
	Diplomi-insinööri Ari Hämäläinen

Hakusanat: tiedonhaku, hakumoottori, sisäverkko
Keywords: information retrieval, search engine, intranet

Yritykset kerryttävät suuria määriä tietoa päivittäisten toimintojensa lomassa ja tallettavat sitä monenlaisiin muotoihin. Tiedon määrän kasvaessa sen haun tehokkuuden on syytä pysyä perässä. Monesti tiedon organisoinnista tulee kuitenkin entistä haastavampaa, ja sen haku vaikeutuu enenevässä määrin. Tällöin tarpeelliseksi tulee etsiä keinoja vähintäänkin tiedonhaun parantamiseksi. Tässä diplomityössä selvitetään suomalaisen yrityksen tiedonhaun kehitystarpeet, etsitään vaihtoehtoisia ratkaisuja tarpeiden täyttämiseksi, sekä otetaan käyttöön yksi näistä vaihtoehtoisista ratkaisuista. Käyttöönotettu sisäverkon hakukone parantaa yrityksen tiedonhakua palvelimen verkkolevyiltä. Tunnistetut kehitystarpeet paljastavat puutteita myös tietämyksen-, tiedon- ja dokumenttienhallinnassa, joiden kehittämiseen yrityksen suositellaan panostavan tulevaisuudessa.

ACKNOWLEDGMENTS

I want to thank Norelco and Ari Hämäläinen for the opportunity to do this thesis, and both Jouni Ikonen and Matti Jaatinen for their guidance.

Most of all I want to thank the people closest to me for their tireless support and compassion during both this thesis work and the whole of my studies.

CONTENTS

SYMBOLS AND ABBREVIATIONS

ACL	Access Control List
API	Application Programming Interface
CAD	Computer-Aided Design
CIFS	Common Internet File System
CLI	Command Line Interface
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DMS	Document Management System
DWG	“Drawing”, file format used by CAD software
GNU	GNU’s Not Linux
GPL	General Public License
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IR	Information Retrieval
IRS	Information Retrieval System
IT	Information Technology
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
MD5	Message-Digest algorithm 5
NTLM	New Technology LAN Manager
OSS	OpenSearchServer
PC	Personal Computer
PDF	Portable Document Format
PNG	Portable Network Graphics
RC	Run Command
RHEL	Red Hat Enterprise Linux
SELinux	Security-Enhanced Linux
SMB	Server Message Block
SSO	Single Sign-On
TCP	Transmission Control Protocol

URL	Universal Resource Locator
VM	Virtual Machine
VPN	Virtual Private Network
WAFFLE	Windows Authentication Functional Framework (Light Edition)
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

1 INTRODUCTION

This master’s thesis describes the research done for identifying the information retrieval challenges of a Finnish company, investigating alternative approaches for solving those challenges and implementing an artefact to alleviate some of them. This section provides an overview of the thesis background, its goals and delimitations, the research methodology used as well as the structure of the report.

1.1 Background

The amount of information companies accumulate over time can be staggering. Successfully managing this information to make efficient use of it is not a trivial matter. It is not only the information that needs to be managed, either, but also the knowledge from which it emerges and the documents or other formats to which it is stored. This section introduces the client company and their current practices in regard to information sharing, and goes over the terminology required to understand the thesis’ subject.

1.1.1 The company and the problem

Norelco Oy is a Finnish company that develops and manufactures electric distribution systems at home and abroad. Like any other company, they produce large quantities of information in a variety of formats, including electronic documents and other data. To store and share this information within the intranet at their home offices, they have a server which is used in various ways. The server’s storage drives are used to store numerous types of files related to the company’s internal operations, and a database located on the server is used to store information about a multitude of items related to the company’s customer processes. A document management system handles the storage of documents attached to the latter.

Information and file sharing between the company’s server and workstations is enabled by Samba, which is a suite of programs for facilitating interoperability between the Windows and Linux/Unix operating systems (Anon 2020a). In effect, the server’s drives are mapped to the Windows workstations as network drives. Additionally, multiple tailored applications are used to access the information contained in the database. In practice, an employee at the company using a Windows workstation connected to the intranet can browse the network drives via Windows File Explorer and view information contained in the database via the applications.

On the network drives, there are various types of documents. The most important ones are the typical office files such as PDF (Portable Document Format) and Microsoft Office files. Naturally, the contents of these files also vary. They range from instructions and templates to brochures, spreadsheets and more. As for the database, information contained in it relates to clients, designs, offers and others.

While information related to key customer processes is stored in the database and the document management system, many of the internal documents remain scattered on the server directories. Employees search for the files on the network drives by navigating from one folder to the next, relying on memorization of the drives' folder structure and naming. The organization of these drives is sometimes unclear, with employees reportedly often struggling to find files they are looking for. Using the Windows search function is at best impractical due to the large number of files and due to there not being a reliable naming scheme in place. While the search functions available on the server itself might be capable enough, their usability for the average employee is limited. Information contained within the database can be searched with various search terms depending on the application being used. There may be significant overlap in what kind of information the different applications present. Quite often an employee has to access multiple applications at once due to not being certain of where a specific piece of information can be found from.

In the initial meeting with the company's representatives, an artefact was described that could be implemented to alleviate these information retrieval issues, at least for the network drives. In their mind, in an ideal situation an employee could search for information by opening a separate search application and inputting an arbitrary search term. The look of the application could be similar to that of the familiar web search engines. Various information about the results could be displayed, for example a snippet of a file's contents, hit highlights and the file's location. The employee could then open the file or the folder it is located in directly from the search window. Many of the drives' files would not be necessary to include in the search system and some of them should even be explicitly excluded. Importantly, file and folder permissions should be preserved in the hypothetical search system. Access to the folders and files is restricted with user accounts, which are identical both on Windows and the server. In principle, each employee has their own account. To log in to the applications, other accounts are used.

1.1.2 Knowledge, information and documents

The concept of information retrieval (IR) is familiar to most people in the modern world largely due to the rise of the world wide web and its search engines. IR as a field of science has its roots in the 1940s. The idea of automatically accessing large amounts of stored data was birthed already in 1945, but the term itself was coined in 1951 (Saracevic 1999; Singhal 2001). Gerard Salton, a long-time leading figure considered by some to be the father of information retrieval, later proposed a definition for the term (Saracevic 1999; quoted in Croft et al. 2015):

"*Information retrieval is a field concerned with the structure, analysis, organization, storage, searching and retrieval of information.*"

In an organizational context, IR is closely related to, or in fact part of, the management of knowledge, information and documents. The exact meaning of these terms can be ambiguous. In their article on the interrelationships between knowledge, information and document management, Chen et al. (2005) recognize this ambiguity and the need for clarifying these fundamental concepts. There are no commonly accepted definitions, but they have formulated one for knowledge as follows:

"*Knowledge is a combination of contextual information and the individual awareness and understanding of facts, truth or information acquired through reasoning, experience and learning. In organizations, knowledge often becomes embedded not only in documents or repositories but also in organizational routines, processes, practices, and norms. […] It is delivered through structured media such as documents, and person-to-person connections. New knowledge is created/acquired through experience, interacting and learning.*"

Knowledge can be tacit (hidden) or explicit. Tacit knowledge is intangible, but may become tangible in the form of information, and vice versa. Knowledge and information affect one another. Information can be seen as a representation of knowledge, and especially for an organization it is also a resource and a commodity. Information can exist in a variety of formats, such as documents and data. Indeed, Chen et al. (2005) succinctly summarize: “A document is the container of written information, and people create [one] by putting information in [it] together with their knowledge”.

The processes of managing knowledge, information and documents within an organization are similarly intertwined. The aim of knowledge management is to make knowledge available to the people of the organization in order to achieve business objectives. It encompasses the management of both information and people. Information management, then, involves effectively managing different information resources and technologies at different levels of the organization. One type of information resource is the paper or electronic document, which belongs to the domain of document management. (Chen et al. 2005)

Understanding these concepts is crucial when determining the goals, delimitations and focus of the research. Though they consist of somewhat distinct activities, implementing changes to one level of management almost inevitably necessitates a more holistic synthetization of them all, which requires investment of another degree entirely.

1.2 Goals and delimitations

The goal of this thesis is to improve the company’s information retrieval. This goal is reached by answering the following research questions:

1. What are the development needs related to the company’s information retrieval?
2. What alternative solutions could fulfill some or all of these needs?
3. How can one of the alternative solutions be implemented?

By gathering qualitative data from communications and interviews with the company personnel, the development needs are identified. The requirements for a solution are formulated based on these needs and the delimitations of the research. These requirements guide the investigation into alternative solutions as well as best practices on implementing them. Based on the findings, an alternative solution is chosen and implemented. In addition, suggestions for further possible development and research are made.

The research has a couple of practical delimitations. Most of the research is done remotely as only a limited number of visits can be arranged on-site. This poses restrictions on data gathering and other practical work. It is for this reason that the research questions purposefully narrow the focus of the research into information retrieval only. The broader concepts of knowledge, information and document management are touched upon, but otherwise largely left out of consideration.

The company’s interests are focused on the practical benefits of the research. It is assumed that as a practical outcome of this thesis, a technological artefact is designed, developed and deployed. Due to the aforementioned delimitation, the scope of such an artefact and the work required to incorporate it has to remain restricted.

1.3 Research methodology

Design science is a research methodology that focuses on the process of developing and evaluating information technology artefacts to solve organizational problems. Artefacts can include instantiations, constructs, methods and models. As one of the aims of this research is to design and deploy an artefact, design science was chosen as the research methodology. The design science research process is illustrated in Figure 1. (Hevner et al. 2004)

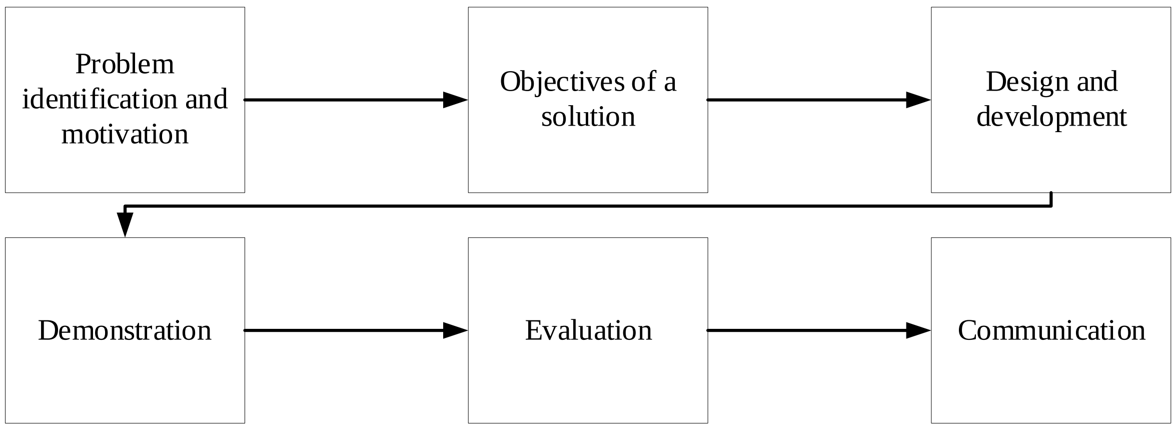


Figure 1. Design science research process (adapted from Peffers et al. 2006).

The steps of the design science research process are carried out in the research as follows:

- The research problem is defined, the value of a solution is justified and preliminary requirements for a solution are identified by communications with the company's representatives.
- The requirements are validated by gathering qualitative data from the company personnel.
- The requirements guide the literature review, which in turn provides background information as well as alternative solutions and guidance for implementing them.
- The artefact is chosen from among the alternatives based on the requirements and delimitations, and its implementation is designed.
- The artefact is deployed, and then evaluated by gathering quantitative and qualitative data from the company personnel.
- The results of the research are communicated via this thesis.

1.4 Structure of the report

The report roughly follows the structure of the design science research process. In this first section, the problem and the motivation for the research was identified, and the preliminary requirements for a solution were described. The second section describes the process of gathering qualitative data for specifying and validating the requirements, and presents its results. The third section presents the results of the literature review into alternative solutions, and the fourth section describes the process of exploring and testing the alternative artefacts as well as designing one for implementation. The fifth section describes the demonstration i.e. deployment of the artefact, and the results of its evaluation are presented in the sixth section. In the seventh and final section the results of the research are presented, the research is discussed and its conclusions are drawn.

2 REQUIREMENTS FOR A SOLUTION

The problem definition formed during the initial meeting and the subsequent communications with the company's representatives had to be validated in order to establish the requirements for a solution. Qualitative data was gathered for this purpose by interviewing the company's employees. The following individuals were interviewed: two from design, three from production and two from sales, seven individuals in total. The representatives themselves represented the administrative department, which was excluded from the interviews. Individual interview sessions were organized, and 30 minutes of time was allotted to each session. The following questionnaire was sent out to each interviewee beforehand:

1) What is your job title and description?
2) What kind of information or files do you regularly need in your job?
3) How often do you search for files from the server or the database?
4) What kind of files do you search for?
5) Specifically, from where and how do you search for the files?
6) How much time does searching for the files take? Do you need assistance from others?
7) Do you yourself produce or add files onto the server or database? What kind of files?
8) Do you have local files on your own workstation that other employees may need? What kind of files?

The questionnaire was designed to help reiterate and verify prior information. It was not designed as a formal, rigid framework for the interviews but instead was meant to help the interviewees to prepare. This also allowed new information to be uncovered organically.

2.1 Interview process and analysis

Almost all of the interviews were conducted at the company's headquarters. One of the planned interviewees was unexpectedly not available, but their stand-in was interviewed in their stead. Six of now eight interviewees, including the one unavailable and their stand-in, had written down answers to the presupplied questionnaire, along with some miscellaneous thoughts. As a result, the time reserved for the interviews was in many cases spent on clarifications and demonstrations in a free-form manner. The audio of five of the six interviews held on-site was recorded. One interview that could not be conducted on the day due to an unforeseen scheduling conflict was arranged to be held remotely at a later date. Seven of the eight total interviewees used their workstations for demonstrating. Video footage was recorded only for the one remote interview. In total, a little over two hours of audio and video footage was recorded. The shortest session lasted around 11 minutes, the longest around 38 minutes and the average session length was around 21 minutes.

The recordings were afterwards transliterated, though not word for word. Only the sections relevant to the topic were included, and individual pieces of information or dialogue were summarized. In vague cases, reasonable assumptions were made about the intent of the interviewee. The findings were compiled and have been summarized here.

Though the work tasks of the three departments differed greatly, the information retrieval practices were virtually identical. Consequently, all of the interviewees expressed very similar thoughts on information retrieval at the company. One key difference was that the interviewees from design especially had more issues with the tailored applications, likely due to the fact that they used them more actively.

Additional types of files the employees use in their daily work were listed. In terms of documents, these ranged from product brochures, standards and instructions to diagrams, tables and images. The file extensions were similarly varied. Other common filetypes in addition to the ones listed previously included JPEG (Joint Photographic Experts Group), PNG (Portable Network Graphics) and DWG (from “drawing”), commonly used by CAD (Computer-Aided Design) software. In terms of information stored in the database, no new types of items were discussed.

Almost all of the interviewees reported that they search for information or files on a daily or at least a weekly basis, sometimes multiple times a day. The retrieval methods and their issues were reiterated. At this point, it turned out that there are multiple applications that access the database, each designed for a different purpose. Due to their naming scheme, they are referred to as the *N-programs*. These applications have been listed in Table 1. The N-programs handle information, including documents, related to business operations, whereas the documents in the network drives are for internal use only. The functions of these applications may overlap, which can cause extra work when trying to find a specific piece of information. For instance, if a client requests a list of components for an electrical center, the center's information may be found in one application, but the components' information in another. While separating these pieces of information may be reasonable, searching for them is highly inefficient. Some information may even have to be retrieved from e-mails. Many other individual issues related to the N-programs were described, especially by one interviewee from the design department. All of these issues related to the general problem of decentralization of information.

Table 1. The tailored applications or N-programs.

Application	**Use case**
NCurmix	Customer relationship management
NAsi	Tracking of design and production
NOffer	Offer system
NProppu	Design, tracking and reporting of production
NDes	Electrical center design
NLask	CAD-based offer calculation

The time taken up by searching varies. Most interviewees reported that it usually doesn't take too long to find what they are looking for, but sometimes it could take "a while". This is especially the case when the information or file is older, more rarely used or otherwise unrelated to the usual daily or weekly tasks. In an outlier worst case, an interviewee reported that they spent an hour a day on average searching for information or files. Some employees reported that they occasionally need help with searching for information, while only a few reported that they are the ones helping.

The amount of time spent searching for information depends on the method. Finding information via the N-programs isn't always a problem despite the discussed issues because the employees have gotten mostly used to the way information is organized. According to some interviewees, the most severe problems arise when having to navigate the network drives and rely on the memorization of file names and locations.

Most of the interviewees reported that they occasionally update or store information on the network drives or database. The conventions for storing files onto the network drives are either non-existent or not documented. Within a single department or team there may be a consensus about where and how to store files, but this is not guaranteed. Especially problematic are the cases where a file's information is relevant to multiple departments, but the naming conventions may differ so much that one department can't find a file stored by another.

Although a couple interviewees reported that there is a company-wide policy in place to prohibit storing non-personal files locally for security reasons, some interviewees reported that they do have such files on their workstations. Usually these files are for personal professional use and possibly specific to one project, but occasionally they may turn out to be relevant to others as well. At such occasions, the files are transferred to the network drives.

The questionnaire didn't include a question on access rights, but some issues related to them were uncovered. Different departments and even individuals have different access rights to folders and files on the network drives. Most of the time employees have access to the information they need, but in relatively rare cases they might not. Usually such problems are resolved easily by either requesting access or the specific file directly from a coworker, but not always.

As for improving information retrieval at the company, the interviewees expressed a desire for a system where information is more centralized and logically organized within the network drives, and more logically presented via the N-programs. Some interviewees felt that having a search interface that would enable searching for files on the network drives would be "extremely useful", while others weren't as certain.

2.2 Results

The aim of the interviews was to validate the information gathered so far and further the understanding of the development needs in regard to information retrieval at the company in order to answer the first research question. The company’s challenges were summarized as follows:

- There are unclear guidelines for storing or organizing documents on the network drives, which makes searching for them troublesome.
- Finding documents on the network drives is reliant on memorization and the tacit knowledge of other employees.
- Some locally stored documents that should be shared with others are not.
- Some documents that should be available to a specific user or group are not.
- In some cases, information contained in the database needs to be pieced together from multiple applications and searches.

It was determined that these challenges were related to all levels of management (knowledge, information and documents) and information retrieval. Though it has been established that the different levels of management are closely intertwined and thus separating these needs into distinct categories may be questionable, development needs were identified and categorized according to these four aspects as follows:

- **Knowledge management**
  - N1: Tacit and explicit knowledge possessed by employees needs to be made available to others.
- **Information management**
  - N2: Workflows and guidelines for managing documents and other information need to be created.
- **Document management**
  - N3: Managing documents needs to be systematic and organized.
- **Information retrieval**
  - N4: Documents need to be searchable via a search interface.
  - N5: Information needs to be presented in a more centralized way via the N-programs, while preserving the required level of access control.

The identified needs N4 and N5 answer the first research question. Due to the practical delimitations of the research, altering the management practices of the company and redesigning the N-programs were deemed infeasible. As such, the investigation into alternative solutions focused on information technology artefacts that could enable searching of documents on the network drive.

3 ALTERNATIVE SOLUTIONS

This section describes the alternative solutions found for fulfilling the company's information retrieval needs. The two types of alternatives are document management systems (DMS) and information retrieval systems (IRS). The architecture of both kinds of systems is described, steps required to implement each types of systems are briefly discussed, and one alternative type is chosen for further investigation.

3.1 Document management systems

Document management is the automated control of documents through each stage in their lifecycle (Cleveland 1995). The number and naming of the stages varies depending on the source, but at least the following can be included: inception or creation, publishing or storage, distribution or retrieval, workflow, and archiving or deletion (Cleveland 1995; van Brakel 2003; Chen et al. 2005). This section will go through each of these stages and the components of a DMS that allow performing the relevant functions. The steps required to implement a DMS will also be presented.

Creation
A DMS may include authoring tools to support document creation. The desktop application in which the document is created may be integrated with the DMS itself to allow storing the document and capturing its metadata directly. Of course, an existing document can also be received from an external source and inserted into the system. (Cleveland 1995; Adam 2007)

Storage
A prerequisite to supporting a DMS is an appropriate underlying infrastructure (Cleveland 1995). This may include servers and workstations connected over a LAN (Local Area Network) or WLAN (Wireless Local Area Network). Storage can be handled in either a centralized or a distributed manner with one or multiple servers. A document repository may contain the documents themselves, while a separate database may be used for the documents' metadata. In the repository, a folder structure may be set up to reflect the organizational structure and/or other classifications. (Sathiadas and Wikramanayake 2003; Adam 2007)

Retrieval

Besides storage, the document repository provides distribution and retrieval functionality. A document may be distributed in different formats, and a number of ways to retrieve them should be available, such as browsing the folder structure as well as basic and advanced search. The data structures, components and functions that enable this kind of search functionality are examined later in this section, when information retrieval and information retrieval systems are discussed in detail. (Cleveland 1995; Adam 2007)

Workflow

Once the document has been created and stored, it has entered the workflow. A workflow is "the movement of a document through a series of steps to achieve a business objective" (Sathiadas and Wikramanayake 2003). A workflow may define a number of actions to take at each step of the document's path. Security may be implemented in the system by allowing users to only view and edit files they have permissions for, and administrative users to set security settings even on a per-folder or per-file basis. Check-in and check-out features may ensure that no more than one person edits a document at any time, and after a document has been edited, the made changes may be tracked by version control. Audit tools may allow authorized users to view the changes that have been made, as well as who made them and when. (Adam 2007)

Archiving or deletion

At the end of a document's lifecycle, it may be determined to have depleted its usefulness to the business objectives, and archived or outright deleted.

Hernad and Gaya (2013) describe a six-step methodology for implementing a document management system:

1. Definition of document requirements
2. Evaluation of existing systems
3. Identification of document management strategies in the organization
4. Design of the DMS
5. Implementation of the DMS
6. Maintenance and continuous improvement of the DMS

Document requirements refer to the types of documents and the workflow that must be established for them to serve the organization and its business objectives. Existing systems both within and without the organization must be evaluated to determine if or how they meet these requirements. After this, an appropriate document management strategy must be identified and adopted, of which four are the most usual (Hernad and Gaya 2013):

- Establishment of principles that set the procedures on document management
- Development of mandatory standards
- Using market IT (information technology) solutions
- Implementation of specific ad-hoc solutions

Considerably different measures can be taken depending on the selected strategy. Regardless, the design of the DMS must be global, that is, it must involve people, processes, tools and technology. The design includes changes to the current systems, processes and practices, the adaption or integration of technological solutions, and the definition of the best way to incorporate these changes. Users are engaged in the design process in order to compare its elements to its requirements. Careful planning is required when implementing the designed system, and besides the implementation, the plan itself has to be developed and maintained to ensure that the most appropriate techniques are used, with minimal disruption caused to the organization. Afterwards, the system's performance must be monitored and corrective measures taken to continuously improve it. (Hernad and Gaya 2013)

3.2 Information retrieval systems

Information retrieval systems, more familiarly known as search engines, can be found everywhere. They can exist as standalone applications or as integrated functionality in others, such as document management systems. According to Croft et al. (2015), the components of a search engine enable two primary functions, namely indexing and querying. These can be further split into subfunctions which are illustrated in Figure 2 and Figure 3, respectively. Indexing consists of acquiring the document text, transforming it and finally creating the index itself. The index is the data structure that enables fast querying, and will be discussed later. Querying comprises the user interaction with the query tool and query processing, as well as ranking the search results and evaluating the engine's performance.

Though not all of these functions are necessarily part of every search engine (Croft et al. 2015), each of them will be examined.

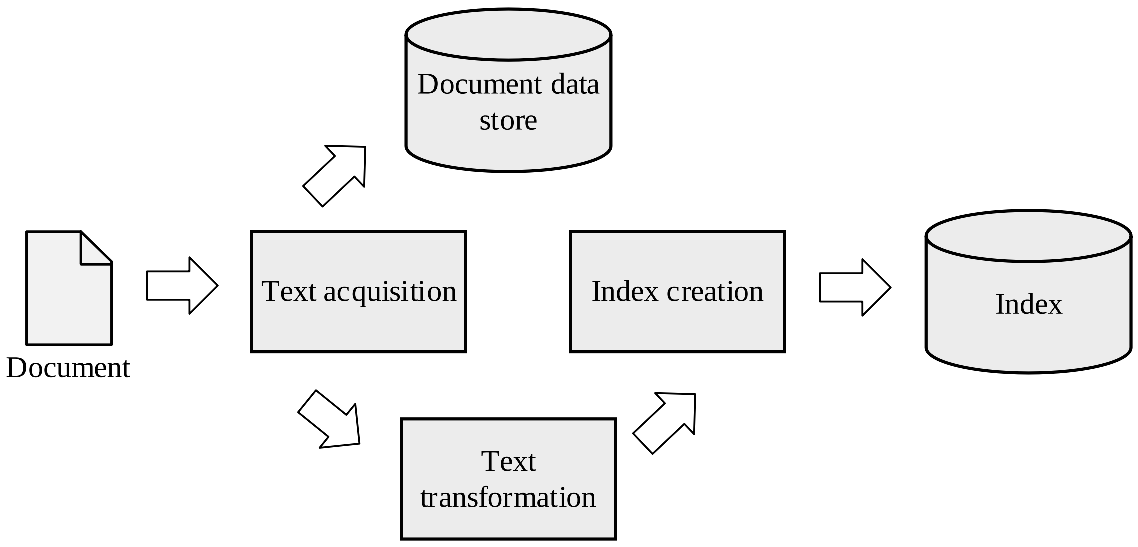


Figure 2. Indexing process (adapted from Croft et al. (2015)).

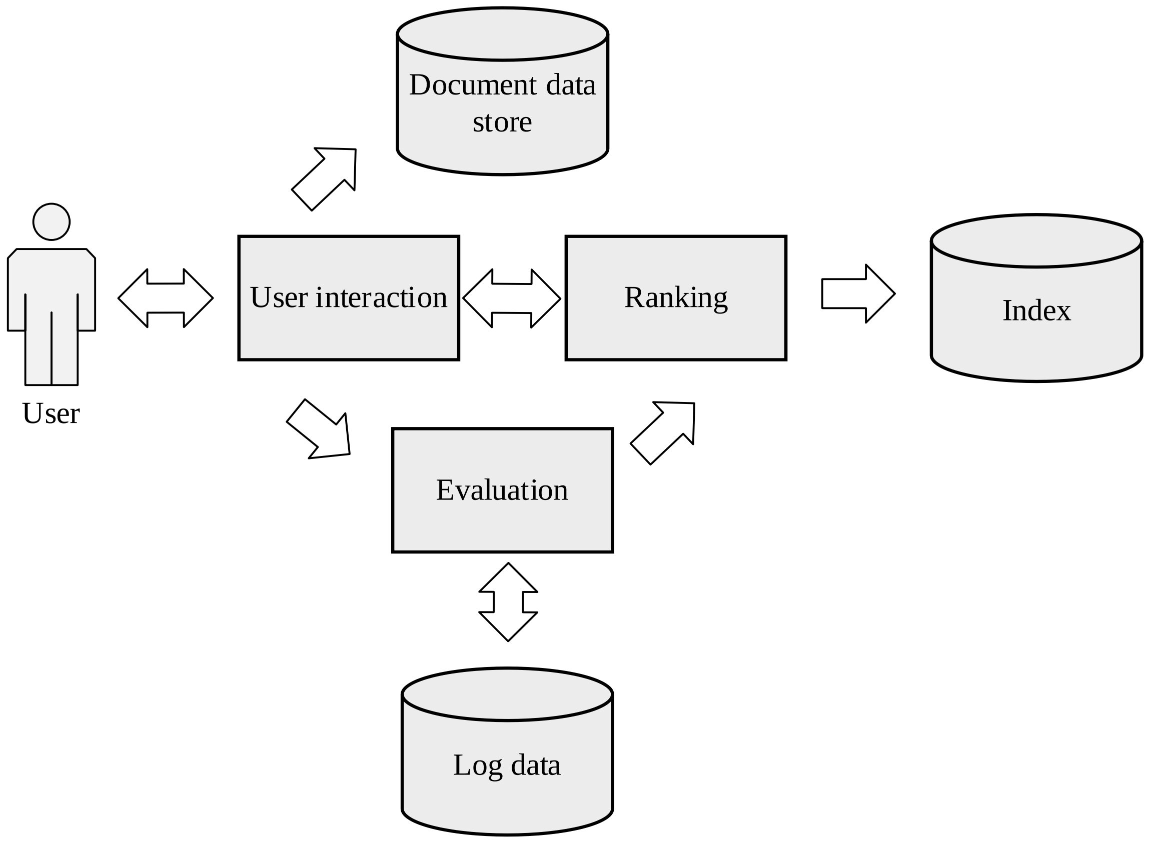


Figure 3. Querying process (adapted from Croft et al. (2015)).

Though most people are familiar with web search engines, these concepts are usually applicable to other types of search engines as well, such as the ones found in desktop or enterprise environments. Indexing other types of content is possible, but only text content will be considered here.

3.2.1 Text acquisition

In order for any text to be indexed, the documents containing it need to be acquired first. The term crawler is often used to describe a component which scours the web, usually to find and index documents, i.e. web pages (Cho and Garcia-Molina 2002). The same idea is applicable to file systems, where instead of traversing web pages through hyperlinks, the crawler navigates the directories and perhaps other computers on the network as well (Croft et al. 2015). To make document acquisition more efficient, multiple crawlers can be deployed simultaneously (Cole 2005). After the initial indexing, the crawlers' job is to update the existing documents and add new ones to the index, preferably in real-time (Cole 2005).

Croft et al. (2015) point out several unique aspects of desktop crawling. Finding documents in a desktop or even an enterprise environment is arguably simpler than in the web, but crawlers face other challenges in these situations. The speed at which changes in the documents are reflected in the index and thus search results is expected to be high, yet it is unreasonable to continuously recrawl the file system. In addition, it wouldn't make sense to store copies of the already local documents like a web crawler would, so the documents need to be loaded into memory and indexed dynamically. Security is a critically important aspect to consider, and access rights to folders and documents should be preserved in the indexing process.

Unlike web pages, desktop data can be quite non-uniform. Since all documents may not be "pure" text files, they have to be converted to a consistent format. This includes the text itself, as well as metadata on the document (Cole 2005). To convert PDF or Microsoft Office files, for instance, external utilities may be needed. The converted documents, along with their metadata and other possible information, can then be compressed and stored for quick access and processing later (Croft et al. 2015).

3.2.2 Text transformation

The raw text in and of itself is not very useful for the purposes of searching. It needs to be processed into index terms, or features, which are used to essentially describe the contents of the documents. These can be not only words but also phrases, names and dates, for example. Usually the text passes through several processing stages, such as tokenization, stopping and stemming, in order to be transformed into index terms (Croft et al. 2015). The transformation process should improve efficiency, but may produce query results that the user might not expect (Baeza-Yates and Ribeiro-Neto 1999).

Tokenization chops the text into individual pieces (Manning et al. 2009). It often produces individual words similar to the final index terms, but in this stage the treatment of special characters, including capital letters, needs to be considered (Croft et al. 2015). Normally words are separated by spaces, but for example it may not be clear whether to treat the words “were” and “we’re” in the same way or not. Tokenization also needs to take the language of the text into account. Goker and Davies (2009) give the example of German and Finnish, in which compound words are common, and external information such as lexicons (collections of known words in the language) are needed to segment or tokenize such words.

Stopping refers to the removal of common words, also called stopwords, from among the tokens. A predefined list of stopwords, similarly to a lexicon, may be used. The size of the index may be significantly reduced by the removal of stopwords, but if the list is too exhaustive, it can even prevent the use of simple search phrases, such as “over there”. (Croft et al. 2015)

Stemming, or suffix stripping, reduces words derived from a common stem into their root form (Büttcher et al. 2010). For example, the words “hand”, “handler” and “handling” could be replaced with the shortest one, “hand”. The stem doesn’t necessarily have to be a recognizable word (Croft et al. 2015). Stemming generally improves recall, but if done too aggressively can decrease precision, similarly to stopping (Kowalski 2011; Manning et al. 2009). In other words, a larger portion of the retrieved documents may be relevant, but fewer of the relevant documents were retrieved in the first place. The language of the text has to be considered as well, since the complexity of different languages’ morphology (formation and structure of words) varies greatly, and for some languages stemming can be ineffective (Croft et al. 2015). The Porter stemmer (Porter 2006) is a popular choice, but according to Porter himself, “there is no point in applying [it] to anything other than text in English.” (quoted in Grehan 2002).

Other aspects of text transformation include extraction of meta-information and classification. Extraction refers to the meta-information being indexed separately from the actual content. Meta-information can include links to web pages, phrases, names, dates, locations and others. Classifiers can identify the type of the document’s content and group and rank the search results accordingly. Notably, they can also detect spam and other non-content. (Croft et al. 2015)

3.2.3 Index creation

The index is arguably the core of the search engine. It is the data structure that enables fast searching, or as Witten et al. (1999) put it, the “mechanism for locating a given term in a text”. There are a number of index types, but the most common is the inverted index, which, according to Zobel and Moffat (2006), is the superior method in terms of retrieval speed. The inverted index in its simplest form, illustrated in Table 2, is constructed of inverted lists, which are mappings “from a single word to a set of documents that contain that word” (Zobel and Moffat 2006). In other words, given a query, the index “tells” which documents contain the query terms. The descriptor “inverted” comes from the fact that it is the opposite of a traditional or a forward index, like one found in a book, which lists all the index terms (and usually their locations) that the document contains.

Table 2. Basic example of an inverted index.

Term	**Document(s)**
lorem	1, 2
ipsum	2
dolor	3, 4, 5
sit	6
amet	7

For the index to be created, statistics of the index terms and the documents related to them need to be gathered. According to Croft et al. (2015), these statistics generally include the counts of index term occurrences, the positions of the terms in the documents, and the document lengths as numbers of tokens. Index terms are weighted to describe their relative importance on a per document basis (Kowalski 2011). The weights can be calculated either at index creation or during querying, but the latter degrades query performance (Croft et al. 2015). There are a number of ways to calculate the weights, though many algorithms are variations of the so-called term frequency-inverse document frequency algorithm, which uses the combination the number of occurrences in a document and the number of documents with that term to calculate weights (Göker and Davies 2009).

Inversion is the part of the process where the index terms and the statistics related to them are used to build the index itself. In other words, the document-term information is transformed into term-document information (Croft et al. 2015). This might at first seem like a trivial task, but the volume of data can easily be too large to be held in memory (Zobel and Moffat 2006). For this reason, disk-based index construction is usually utilized, while in-memory construction is reserved only for relatively small collections (Büttcher et al. 2010). Inversion needs to be done efficiently not only at index creation, but also when the index updated (Croft et al. 2015).

The index is usually compressed. Multiple indexes can be distributed across several computers to enhance query performance. The indexes can be replicated or distributed for a subset of documents or terms, which can reduce communication delays and enable parallel processing, respectively. (Croft et al. 2015)

3.2.4 User interaction

Once the index has been created, the user can query it. For this the user needs to be provided an interface for input. A parser will process the user’s input according to a specific query language. In this process the query terms will need to be transformed (similarly to the original text) so that they can be compared to the index terms (Kowalski 2011). Advanced query processing may include spell checking, suggestions and other analysis, but these are more often seen in web search engines than in desktop or enterprise environments (Croft et al. 2015). After the query has been processed, the results are displayed in a ranked order. The results may include any information stored about the results, such as snippets of their contents.

3.2.5 Ranking and evaluation

For the results to be displayed in order of relevance, they need to be ranked. The ranking algorithm (or whether one is used at all) is based on the retrieval model used, but in any case the score given to a document essentially reflects its relevance to the given query. This is achieved in many retrieval models by giving weights to both the query and index terms. There are numerous different retrieval models, but they won't be covered in this thesis. Suffice to say that a retrieval model is a formal representation of the process of matching a query and a document. (Croft et al. 2015)

Once a search engine is up and running (and being used), its performance may be monitored, evaluated and improved. According to van Rijsbergen (1979) and Croft et al. (2015), the two key qualities of a search engine are effectiveness and efficiency. The engine is effective when it retrieves the most relevant documents possible, and efficient when it does this in as little time as possible. Numerous metrics as well as logging and analyzing the users' interactions with the engine can be used to evaluate the engine's effectiveness and efficiency.

The effort that goes into deploying a search engine largely depends on the engine. As Bancilhon (1999) puts it, "some search engines are literally 'turnkey'", while others can be rather complex to implement. Deploying a search engine can be as straightforward as downloading, running and configuring a server instance. Some engines are actually used as platforms for building bespoke search engines or adding search functionality into other applications. Implementing a search engine with these platforms may require a substantial amount of configuration and technical know-how, but the control over the end result is much greater.

3.3 Conclusions on alternative solutions

In terms of improving information retrieval only, both document management and information retrieval systems are suitable artefact candidates. The company has a DMS in use, but it is currently not utilized for the internal documents on the network drives. A DMS may offer a more holistic solution that addresses the underlying document management issues on the network drives as well, but planning and executing its implementation demands a significant amount of effort. In the context of this research and its delimitations, this was deemed too large an undertaking.

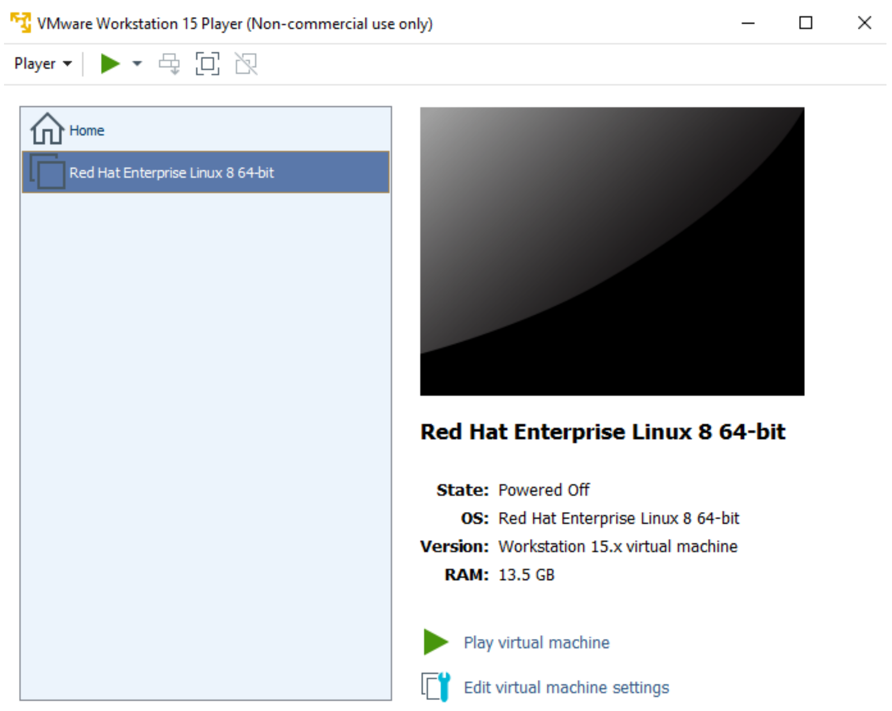
An information retrieval system consists of numerous components and algorithms, which could in theory be implemented in a bespoke search engine. In reality, building a search engine is an enormous project, with many existing ones having been in development for several years. The company was interested in exploring existing alternatives that could provide a cost-effective solution in the present. As such, the investigation proceeded with exploring existing search engines and their feature sets, which were then compared with the requirements in order to find suitable candidates for testing and eventual implementation.

4 TESTING AND DESIGN

In order to effectively test the search engines prior to choosing one for implementation, an environment that resembled the real one as closely as possible was set up. With the relevant components present in the environment, the functionality of the search engines could then be simulated. Candidate search engines were explored, compared to each other and with the requirements, and tested. One engine was chosen for implementation and its deployment was designed. This section describes the testing and design process.

4.1 Setting up a testing environment

The Red Hat Enterprise Linux (RHEL) Server operating system used on the company's server is a commercial product. The developer Red Hat offers a free 30-day trial for RHEL 8, as well as a completely free developer version as a disc image file (Red Hat Inc. 2020). This file may be used to install the operating system on a real or a virtual machine (VM). To create a virtual machine, the free version of VMware Workstation Player was used (VMWare, Inc. 2020). Basic settings for hardware simulation, such as the amount of memory available and the type of network connection used, were set.


VMware Workstation 15 Player (Non-commercial use only)
Player
Home
Red Hat Enterprise Linux 8 64-bit
Play virtual machine
Edit virtual machine settings

Red Hat Enterprise Linux 8 64-bit

State: Powered Off
OS: Red Hat Enterprise Linux 8 64-bit
Version: Workstation 15.x virtual machine
RAM: 13.5 GB

Figure 4. VMWare interface.

After launching the virtual machine from the host Windows through VMWare as seen in Figure 4, the operating system was installed from the disc image file. In order to enable the installation of packages, the free subscription was attached to the installation. At this point the virtual RHEL was ready for Samba to be installed. This was done with the following commands (a dollar sign indicates a CLI (command line interface) command). These commands install the packages for Samba and the optional Samba client, start and enable relevant services so that they run at startup, and ensure that traffic through the necessary ports is allowed (Docile 2019).

```
$ yum install samba samba-client
$ systemctl enable --now {smb, nmb}
$ firewall-cmd --permanent --add-service=samba
$ firewall-cmd –reload
```

By default, SELinux (Security-Enhanced Linux) may interfere with external machines trying to access a Samba share. This can be circumvented in a number of ways. For the virtual machine, SELinux can be temporarily disabled with the command (Mutai 2019)

```
$ setenforce 0
```

Access to Samba shares is allowed or restricted according to usernames and group names. The users were created on RHEL and added to Samba explicitly. To do this, the following commands were used (Anon 2020b). Placeholder parameters are signified with square brackets (for example [parameter]). The first command creates the user without a home directory and prevents them from logging in. This may be useful in the case that the accounts are only used for authenticating access to the Samba shares.

```
$ useradd -M -s /sbin/nologin [username]
$ passwd [username]
```

Adding the user to Samba and enabling that user:

```
$ smbpasswd -a [username]
$ smbpasswd -e [username]
```

Creating a group and adding the user to that group:

```
$ groupadd [groupname]
$ usermod -aG [groupname] [username]
```

Next a share folder was created, and an access control list (ACL) was defined for it. Various flags can be used to add or remove access rights for specific users or groups. Default rights can be set, so that any files or folders created within the folder inherit those rights. ACLs can be checked with the command *getfacl*.

```
$ mkdir -p [path]
$ setfacl [flags] \
user/group/other:[username/groupname/empty]:[rights] [path]
$ getfacl [path]
```

As an example, to give the group *other* (i.e. everyone else but the owning user and group) full permissions to the folder *public*, and to make new files and folders inherit those rights, the following commands can be used:

```
$ setfacl -d -m o::rwx public
```

And the output from *getfacl*:

```
# file: public
# owner: search
# group: search
user::rwx
group::rwx
other:rwx
default:user::rwx
default:group::rwx
default:other::rwx
```

To create the Samba shares themselves, the file */etc/samba/smb.conf* was edited. In Listing 1 an example of *smb.conf* can be seen, which defines two shares with minimum configuration. The *global* section defines parameters that apply to Samba as a whole (here square brackets signify sections or shares). Only connections from the local network are allowed. The first share, *public*, is open to all users (though not guests), while the second, *notpublic*, is only accessible to users who belong to the group *notpublic*. There are dozens if not hundreds of parameters that can be set in *smb.conf*, but these will not be discussed.

Listing 1. Example of *smb.conf*.

```
[global]
    hosts allow = 127. 192.

[public]
    path = /home/user/Samba/public
    read only = no

[notpublic]
    path = /home/user/Samba/notpublic
    read only = no
    valid users = @notpublic
```

Each time *smb.conf* is edited and saved, the configuration needs to be reloaded with the command

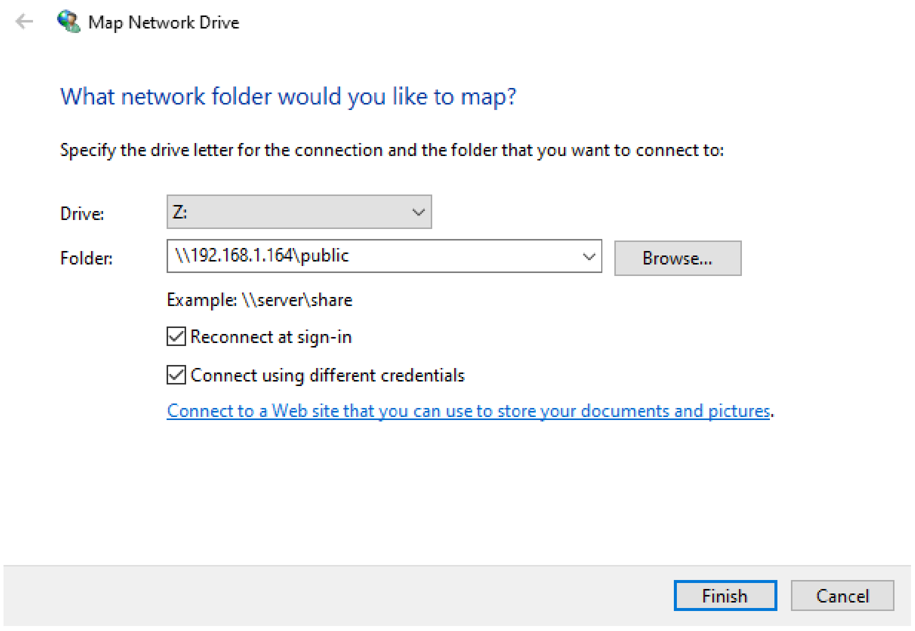
```
$ smbcontrol all reload-config
```

To access the Samba shares from the host Windows, insecure guest logons had to be enabled, even if a user account was used to log in. This was done via Windows' Local Group Policy Editor. In order to test multiple user accounts on the same host machine, multiple hostnames needed to be mapped to the same server IP (internet protocol) address. This was done by modifying the file *C:\Windows\System32\drivers\etc\hosts*. For example, to map the hostnames *public* and *notpublic* to the IP address 192.168.1.164, the following lines can be added:

```
192.168.1.164 public
192.168.1.164 notpublic
```

On the host Windows, the Samba shares were mapped as network drives using the defined hostnames as seen in Figure 5. For convenience, the hostnames referred to the user that the mapped drive was accessed with, though any credentials could have been used. The mapping had to be done to a specific share (*public* or *notpublic*), but different shares could be accessed (if available) with the same credentials by typing the appropriate URL (Uniform Resource

Locator) into the Windows Explorer address bar. After the mapping, the drive is accessible from Windows File Explorer's This PC (Personal Computer) section, as seen in Figure 6.


Map Network Drive

What network folder would you like to map?

Specify the drive letter for the connection and the folder that you want to connect to:

Drive: Z:

Folder: \\192.168.1.164\public

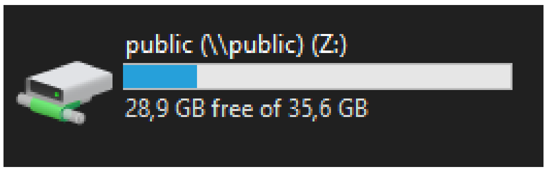
Example: \\server\share

☑ Reconnect at sign-in

☑ Connect using different credentials

Connect to a Web site that you can use to store your documents and pictures.

Figure 5. Mapping a network drive.


public (\\public) (Z:)

28,9 GB free of 35,6 GB

Figure 6. Mapped network drive.

When a share is accessed through the mapped network location, any new files or folders are created with the used credentials, and inherit any default ACLs. Parameters in *smb.conf* can be set to further tune default permissions. With the Samba shares up and running, search engines could be installed and tested.

4.2 Exploring, comparing and testing search engines

In this section available search engines are explored and their features are compared to the requirements. To narrow down the search, functional and non-functional requirements for the engines were identified based on three things: the problem definition and other communications with the company's representatives, the established requirements based on the company’s needs, and the list of factors by Bancilhon (1999) below. The search engine requirements are listed in Table 3, and were approved by the company’s representatives. The requirements are prioritized according to the so-called MoSCoW method. The priority levels (Must have, Should have, Could have, Won’t have) are denoted with the letters M, S, C and W. The viable search engines that were found and matched at least most of the requirements are listed in Table 4. Each engine is described briefly.

Bancilhon (1999) lists the following factors to consider when choosing an intranet search engine:

- Server dependability
    - What platform and type of server will the engine run under?
    - Are there multiple servers or just one?
- Types of documents to be indexed
- Types of searches
- Security
    - How are access rights implemented in the intranet?
- Platform dependability
    - On which platforms should the engine be able to index documents?
- Search interface
- Speed
- Costs
- Indexing and timeliness
    - How up-to-date should the index be?
- Accuracy and relevance
- Administration and degrees of control
    - How much administrative functions does the search engine provide?
- Ease of implementation
- Disc space and directory consideration
    - How much memory does the engine require?
    - How much disk space does the index require?
- Size of organization and expected importance of intranet
    - How decentralized is the organization and its information?

- How fault-tolerant should the engine be?

- Reporting functions
  - Should the engine provide logs of its usage, performance or other aspects?
- Employee training
  - How difficult is the search interface to use for an employee?
  - How much resources are available for training personnel?
- Convincing management
  - How reluctant is the company's management to implementing the engine?

These factors were considered together with the aforementioned aspects when identifying the functional and non-functional requirements.

Table 3. Functional and non-functional search engine requirements.

ID	**Name**	**Priority**	**Description**
R1	Open source	M	Preferably open source software to make testing easier and early commitment to any one solution unnecessary, and to keep costs low.
R2	RHEL	M	Support for Red Hat Enterprise Linux (or Linux in general).
R3	Enterprise	S	Built for enterprise search purposes.
R4	On-premises	M	Deployed on-premises.
R5	File formats	M	Support for indexing contents of various file formats.
R6	Setup and forget	C	Easy installation, and minimal configuration and maintenance required during and after deployment.
R7	Ready UI	S	Readily usable (and optionally customizable) UI accessible from any modern browser, including mobile ones.
R8	Access control	M	Support for login functionality and handling access rights.
R9	Real-time	M	Real-time or almost real-time indexing.
R10	Administration	S	Includes administration tools for monitoring, configuration, and others.
R11	Documentation	S	Sufficient documentation to aid deployment and maintenance. On-going development is a plus.

R12	Queries	S	Supports multiple types of queries, for example fuzzy and wildcard queries.
R13	Results	S	Results view includes content snippets, hit highlighting and others.
R14	Finnish	C	Support for the Finnish language.

Table 4. Matches between the search engines' features and the requirements.

	Requirement													
Search engine	Open source	RHEL	Enterprise	On-premises	File formats	Setup and forget	Ready UI	Access control	Real-time	Administration	Documentation	Queries	Results	Finnish
Ambar	~	~	✓	✓	✓	✓	✓	?	✓	?	~	✓	✓	✗
Apache Solr	✓	✓	✓	✓	✓	✗	~	✓	✓	✓	✓	✓	✓	✓
Datafari	~	~	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	?
Elasticsearch	~	✓	✓	✓	✓	~	✗	~	✓	✓	✓	✓	✓	✗
Everything	✓	✗	✗	✓	✓	✓	✓	?	✓	?	✓	✓	✗	?
OpenSearchServer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	~	✓	✓	✓
Open Semantic Search	✓	?	✓	✓	✓	✓	✓	?	✓	✓	✓	~	✓	~
Searchblox	~	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yacy	✓	✓	✓	✓	✓	~	✓	?	✓	✓	~	✓	✓	?

Ambar

Ambar is based partially on Elasticsearch and is deployed with Docker (Docker Inc. 2020). A paid enterprise version of Docker is required for RHEL. It’s unclear how access control may be implemented with the engine, and its development doesn’t seem particularly active. (Ambar LLC 2020)

Apache Solr

Apache Solr is a popular search platform that is highly extensible and can be tailored to many use cases. Many other search engines, including some listed here, are based on Solr. While it provides all of the desired features, deploying and configuring it may be too complex while other less complicated engines may produce similar results. The certainty of continued support and development is a definite advantage. (Apache Solr Software Foundation 2020)

Datafari

Datafari is based on Solr which, akin to Ambar, utilizes Docker. Although otherwise a strong candidate, a paid enterprise edition is required for RHEL. (France Labs 2020)

Elasticsearch

Similarly to Apache Solr, Elasticsearch seems to be more often used as a platform for other search engines. Its feature set is exhaustive, but some critical components are restricted to paid editions. (Elasticsearch B.V. 2020)

Everything

Everything is perhaps the most straightforward search engine to set up on this list. While it supports indexing network drives, it is only available for Windows. The index would have to be stored locally on each workstation, which is not ideal. Another disadvantage is that the engine cannot index file contents. (Carpenter 2020)

OpenSearchServer

OpenSearchServer is an open source search engine with a complete feature set and a simple setup process. Though development is somewhat active with the website being updated in preparation for the upcoming new version, the engine’s documentation is severely lacking. Nevertheless, it is still a strong candidate to consider. (OpenSearchServer, Inc. 2020a)

Open Semantic Search

Open Semantic Search is based on both Apache Solr and Elasticsearch, and its functionality is comparable to Ambar and Datafari. However, its support for RHEL is unclear. (Mandalka 2020)

Searchblox

Searchblox is also based on Elasticsearch. It’s simple to set up and a free license is available, though it can only be used for one collection. This means that a single index of only 10 000 documents can be created, which is a severely limiting factor. (SearchBlox Software, Inc. 2020)

Yacy

Yacy is an open source engine that is focused on decentralization and P2P (peer-to-peer) networks for web indexing, but also usable as an intranet search engine. It almost has a complete feature set but its interface is confusing, it doesn’t seem to support access control, and its documentation is practically nonexistent. (Christen 2020)

Not all of the search engines listed in Table 4 were properly tested in the virtual machine due to the fact that either installing them proved unsuccessful or significant problems were encountered during setup. Ambar and Datafari were successfully installed but indexing a network share turned out to be impossible. OpenSearchServer, SearchBlox and Yacy were successfully used to index a Samba share. SearchBlox’s limit of 10 000 documents proved prohibitive, but the other two engines were showcased to the company’s representatives. OpenSearchServer was chosen for further testing and ultimately implementation.

4.3 Designing the OpenSearchServer implementation

Once OpenSearchServer was chosen for further testing, its implementation was designed within the test environment. This section describes the main features of the engine and how they were utilized in the implementation. This section will not serve as documentation for the engine’s interface or each of its components’ settings, but will instead focus on the design choices on a higher level.

OpenSearchServer (OSS) is an open source enterprise-class search engine developed by OpenSearchServer Inc. and licensed under the GNU (GNU’s Not Unix) GPL (General Public License) v3 (Free Software Foundation, Inc. 2020). It enables the integration of search capabilities into existing applications and can also function as a standalone search application via its web user interface. It requires only that the system has a Java runtime installed (version 7 or later) and can be downloaded as a tar.gz archive and unpacked anywhere. Scripts are provided for starting and stopping the server, which runs on the TCP (Transmission Control Protocol) port 9090 by default. The web user interface can be accessed from another system using any modern browser. (OpenSearchServer, Inc. 2020d; OpenSearchServer, Inc. 2020b)

The design of the OSS implementation started with the company's representatives describing the desired end result. They expressed that the interface should resemble the ones found in popular web search engines, with simplicity and clarity as the main priorities. It was determined that the interface should include the following elements:

- Company logo
- Login page and logout button
- Search bar with autocompletion
- Search results
    - Exact filename, which links to the file on the network drive
    - Path to the folder, which links to the folder on the network drive
    - Snippet of content with hit highlighting
    - Date
- Filters
    - Date
    - File extension
    - Language
    - Share
    - Type (directory or file)
- Sorting
    - Relevance
    - Date (ascending)
    - Date (descending)

The admin interface of OSS provides functionality for defining all the necessary components for customizing the search engine. These include the indexes, the index schemas, the analyzers, the crawlers, the query template, the renderer, the schedules and the authentication.

4.3.1 Indexes

Setting up an OSS instance starts off with creating an index. An index can be given a name, and a couple of templates are provided for a quick setup. These include templates intended for web or file crawling, as well as for storing user credentials.

Two indexes were created; one main index for the documents using the file crawler template, and another index using the credentials template. The following sections mostly describe components defined for the main index.

4.3.2 Schema

A schema describes the structure of a database and the information it contains. An index is essentially a database, and its schema lists the fields that are stored for each document. These fields contain pieces of information like the document's name, path and extension, as well as its content, date, permissions and others. The index fields' names can be arbitrarily chosen, so they have to be mapped to the real file fields. For example, the document's filename may be mapped to the *fileName* field.

In OSS, the schema defines whether a specific field's contents are indexed or not, and how. Prior to indexing, the field's contents can be processed with a so-called analyzer. A field's original contents can also be stored separately. A list of terms, such as a document's access rights or contents, can be stored as a term vector. The term vector's offset information can be stored as well, which enables the use of snippets. A field's contents can also be a copied off another field. Copying is done prior to processing, so the contents are equal before they are processed by any analyzers.

The schema needed to at least include the fields that are displayed in the search results, the fields that the queries are executed within, as well as the ones used for filtering and sorting. Table 5 lists all of the indexed fields and their properties that were included in the schema.

No new mappings to the real file fields had to be added. Some of the fields are quite self-explanatory, but those that are not are further explained.

Table 5. Schema of the main index.

Field name	**Stored**	**Term vector**	**Analyzer (copy of field)**
lang	No	No	
title	No	No	TextAnalyzer
content	Yes (compress)	Yes (offsets)	TextAnalyzer
url	No	No	
fileName	Yes	No	TextAnalyzer
autocomplete	No	No	AutoCompletionAnalyzer (copy of *fileName* and *content*)
directory	No	No	
crawlDate	No	No	
fileSystemDate	No	No	
fileExtension	No	No	
fileType	No	No	
fileSize	No	No	LongAnalyzer
userAllow	No	Yes	AccessAnalyzer
userDeny	No	Yes	AccessAnalyzer
groupAllow	No	Yes	AccessAnalyzer
groupDeny	No	Yes	AccessAnalyzer
winURL	No	No	WinURLAnalyzer (copy of *url*)
winDir	No	No	WinURLAnalyzer (copy of *directory*)
share	No	No	ShareAnalyzer (copy of *url*)

The fields *title* and *fileName* may seem like they represent the same thing, but in fact the former refers to the metadata title of the document, which may be different from the actual filename. The *fileName* field is also stored as is to allow displaying the exact filename in the search results.

To allow displaying snippets of the documents' original contents in the search results, the *content* field is indexed as a term vector with offset information, as well as compressed and stored in its original form. The *autocomplete* field is copied from the *fileName* and *content* fields and processed separately.

The *url* field is the file's path. The *winURL* field is the same path, but in Windows format and with the IP address replaced with the name of the mapped network drive. The two formats compare as follows:

```
smb://192.168.1.1/share/folder/file
file:\\mappedshare\share\folder\file
```

The fields *directory* and *winDir* follow the same logic. Both *winURL* and *winDir* were used in the search results as hyperlinks for the filename and filepath, respectively. Practically every employee and workstation uses Windows, so including links compatible with other operating systems wasn't necessary.

The names of the *fileExtension* and *fileType* fields can be misleading. The former is the file's format, while the latter states whether the document is a file or a directory. The fields *userAllow*, *userDeny*, *groupAllow* and *groupDeny* contain the allowed or denied users or groups as lists of terms. In reality, the *deny* fields are rarely if ever used but are included nonetheless.

4.3.3 Analyzers

The content of some fields (such as *lang*) can be indexed as is, though for other fields the contents may need to be processed in order to save disk space and make querying more efficient. Analyzers can be defined to process the fields' contents prior to indexing. These can be defined for each desired language separately. An analyzer first tokenizes the field's contents and then (optionally) runs the tokens through various filters. This is the step of the process where stop words can be removed and stemming can be performed. Custom lists of stop words for each language can be added. An arbitrary number of predefined filters with numerous parameters can be added to each analyzer. The end result of an analyzer depends on what kind of information should be stored for a specific field.

Five distinct analyzers were defined, and one of these analyzers was defined for a couple of languages; Finnish and English. The analyzers and examples of their input and output are listed in Table 6. Each analyzer is used both during indexing and querying, meaning that a user’s query is processed in the same way as a field’s content. Only two kinds of tokenizers have been used for all of the analyzers. The standard tokenizer is used for text. It strips the text of special characters like commas, periods and white space, and treats each word as a separate token. The keyword tokenizer, on the other hand, treats the whole input as a single token and does not process it in any way. This is useful for fields where the content is not typical text.

Table 6. Created analyzers.

Analyzer	**Tokenizer**	**Input example**	**Output example**
Access	Keyword	unix user\username	username
AutoCompletion	Standard	this is text	this is, this is text, is text, text
Share	Keyword	smb://ip/public/file.txt	public
Text	Standard	this is text	tex, text
WinURL	Keyword	smb://ip/public/file.txt	file:\\mappedshare\public\file.txt

The *AccessAnalyzer* simplifies how the names of users and groups are presented. The keyword tokenizer is used, and the names are stripped of any special prefixes using regular expression (regex) replace filters. Special users and groups that are extracted by the file parsers but are not needed are excluded with another regex filter.

The *AutoCompletionAnalyzer* is used to generate suggestion snippets from the files’ names and contents. The standard tokenizer is used. The tokens are transformed into lower case, any file extensions are removed, and a filter is used to generate shingles from them. Shingles can be thought of as varying groups of words. Lastly, stop words are removed so that there are no suggestions like “this is a”.

The *ShareAnalyzer* is used to extract the share name from the document’s URL. The keyword tokenizer and two regex replace filters are used to achieve this.

The *TextAnalyzer* is perhaps the most important analyzer. It transforms the contents of the three most important fields (*fileName*, *content*, *title*) into a form that saves disk space and is more efficient to query. A separate analyzer has been defined for each language used, including an “undefined” language which is used when a parser cannot determine the document’s language. The standard tokenizer separates the words into tokens, and the tokens are filtered several times. The exact filters depend on the language. First, the tokens are transformed into lower case. Stop words are removed using custom stop word lists. Umlauts like *Ä* and *Ö* are transformed to *A* and *O*. Either a snowball or a stem filter is used to stem the tokens. Lastly an edge n-gram filter is used to produce partial tokens of varying length, similarly to the shingle filter, but for each individual word. This allows the end user to search with only partial or even mistyped search terms.

The *WinURLAnalyzer* uses the keyword tokenizer and two regex replace filters to transform both the document and the folder paths to Windows format.

The *LongAnalyzer* seen in Table 5 is a predefined analyzer that transforms the file size information into a format used internally by OSS.

4.3.4 Crawler

OSS supports crawling of various sources including the web, databases and mailboxes, as well as local and remote file systems. Depending on the crawl target, a number of settings can be set to include and exclude resources. The crawl process can be set to run once or indefinitely and scheduled to run automatically. OSS includes a multitude of parsers that are able to handle most of the popular file extensions during crawling and indexing, such as PDF and the Microsoft Office formats. The parsers’ settings can be changed but this is rarely necessary.

In the case of crawling a file system (including a network drive), so-called “locations” are defined. Locations include information like the root of the crawl, exclusion patterns and the type of permissions that are extracted from the documents. A single location was defined as for each Samba share that was indexed. The crawler needed to be provided credentials which it used to access the share. To avoid having to create multiple locations with different credentials for each share, a “super user” with access to all shares and their files was used. An example of a location definition for a Samba share is as follows:

- Type: SMB (Server Message Block)/CIFS (Common Internet File System)
- Username: superusername
- Password: superuserpassword
- Security permissions: File & share permissions
- Host: 192.168.1.1
- Path: /share/
- Exclusion patterns: */notindexedfolder/*

The crawl process can be started as soon as the schema, analyzers and crawl locations have been defined. If authentication is going to be used, the relevant fields need to have been included in the schema.

4.3.5 Query template

A query template defines what happens when an end user enters a search term into the search bar. Most importantly this includes which fields the query is executed within and which fields are returned. Queried fields can be given different weights, so for example the document’s name can be given a greater importance over its content. The returned fields essentially comprise the information displayed for each search result. Fields that are going to be used for snippets and filters are also defined in the query template.

The queries can be executed with the terms being treated as one of the following (OpenSearchServer, Inc. 2020b):

- Pattern
    - No processing is done to the search terms.
- Term
    - Special characters are removed, and the query is executed with each term individually.
- Phrase

    - Special characters are removed, and the query is executed with the phrase as a whole.
- Term & phrase
    - Combination of the previous two.

A single query template was defined. The searched or queried fields were *title*, *content*, *fileName* and *url*. Within each field, the query is executed both as separate terms and a single phrase. The fields *title* and *fileName* were given a weight twice that of *content* and *url*. Each field was given a phrase slop value of three, which made it possible to match phrases even if their terms are in a slightly different order.

The returned fields comprise the information that is displayed for each search result. These are *fileName*, *winURL*, *winDir*, *content* and *fileSystemDate*. though *winURL* is not displayed by itself but instead used as a hyperlink for *fileName*. *Content* is the only snippet field. The filter fields are *lang*, *fileExtension*, *share* and *fileType*.

4.3.6 Renderer

A renderer defines a search interface, including its appearance and the information it displays. A specific query template can be attached to a specific renderer. Basic renderer templates using basic HTML (Hypertext Markup Language) and the Bootstrap framework (Bootstrap Core Team 2020) are provided. The styling of the interface can be further altered with CSS (Cascading Style Sheets). A header, a footer and labels as well as filters and sorts can be defined within the renderer settings. The renderer also determines the authentication type used, if any.

A single renderer was defined, and the created query template was attached to it. The Bootstrap HTML template was used, basic text labels were defined and a test logo was included in the header with the following HTML element:

```
<img src="./images/logo.jpg" height="80" margin-bottom="10">
```

The renderer fields, listed in Table 7, were almost the same as the returned fields in the query template. The *content* field was defined as a snippet. The *winURL* and *winDir* fields were used as links. Each of the fields used a different CSS class for styling. Dates were formatted as *dd.MM.YYYY*.

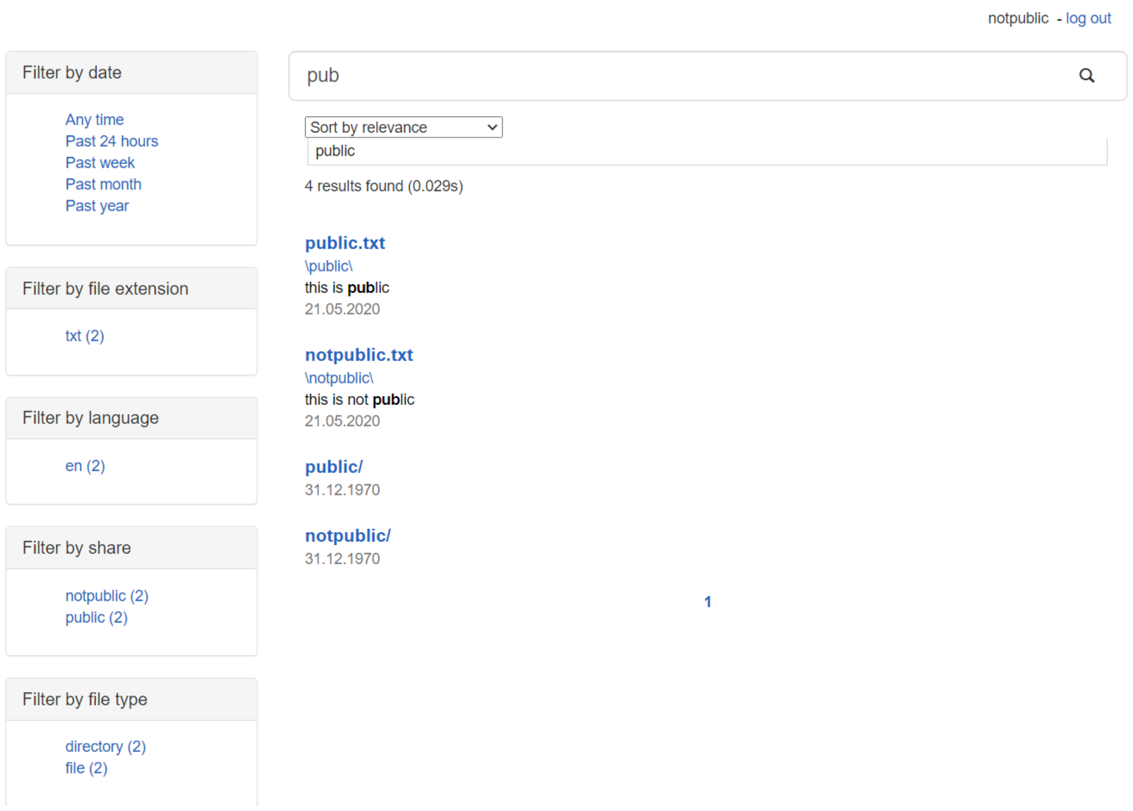
Table 7. Renderer fields and their settings.

Field/Snippet	**URL field**	**URL decode**	**CSS class**	**Widget type**
fileName	winURL	False	filename	Text
winDir	winDir	True	location	URLwrap
content		False	content	Text
fileSystemDate		False	small-text-muted	Datetime

The same filters defined in the query template were added to the renderer, and three different sorts were added; relevance, date (ascending) and date (descending). Most of the default styles were kept as is, but a couple styles were altered and a couple new classes were defined as seen in Listing 2. The aim was to make the color scheme resemble the one used by the company. The final look of the interface has been visualized in Figure 7.

Listing 2. Modified and added CSS styles.

```
a {
 color: #2065C0;
}

.ossfieldrdr3 {
 color: black;
}

#oss-header{
 margin-bottom: 20px;
}

.filename {
  font-weight: bold;
  color: #2065C0;
}

.content {
  color: black;
}

.location {
  color: blue;
}
```


notpublic - log out

Filter by date

Any time
Past 24 hours
Past week
Past month
Past year

Filter by file extension

txt (2)

Filter by language

en (2)

Filter by share

notpublic (2)
public (2)

Filter by file type

directory (2)
file (2)

pub

Sort by relevance
public

4 results found (0.029s)

public.txt
\public\
this is **pub**lic
21.05.2020

notpublic.txt
\notpublic\
this is not **pub**lic
21.05.2020

public/
31.12.1970

notpublic/
31.12.1970

1

Figure 7. Designed search interface.

4.3.7 Authentication

To authenticate end users connecting to the search interface and only display results they have access to, a couple of things are needed. First, the schema needs to include fields for the allowed and denied users and groups, and those fields need to be mapped to the corresponding real file fields. Second, authentication needs to be enabled in the renderer. OSS provides the following authentication types:

- HTTP (Hypertext Transfer Protocol) header
- NTLM (New Technology LAN Manager)/NTLM with login
- Index with login
- WAFFLE (Windows Authentication Functional Framework (Light Edition))/SSO (Single Sign-On)

As seen in Table 5, the fields for access rights were added to the schema and the *AccessAnalyzer* was set to process the extracted rights into a form understood by OSS. In addition, authentication was enabled in the renderer's settings. The simplest form of authentication available was used; an index containing the users' credentials. The schema of the credentials index can be seen in Table 8. The *CryptAnalyzer* encrypts the password during indexing. It appears to use the MD5 (message-digest algorithm 5) hash function, and accepts a custom salt as input.

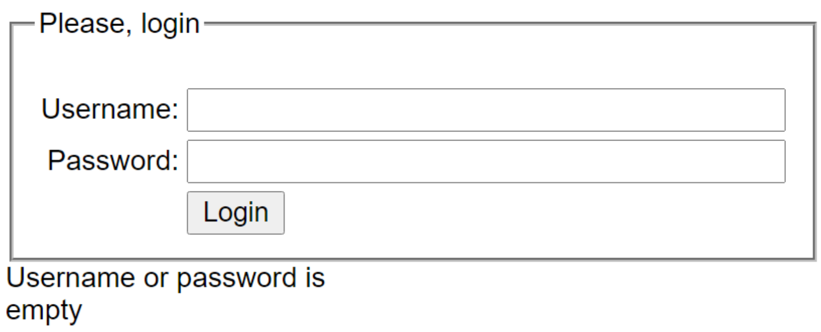
Table 8. Credentials index schema.

Field name	**Stored**	**Term vector**	**Analyzer (copy of field)**
username	No	No	
password	No	No	CryptAnalyzer
groups	No	Yes	

OSS has two kinds of users; those that are inserted into the credentials index and can log in to the search interface, and those that can login to the admin interface itself. Both kinds of users are needed to access a defined renderer, i.e. search interface. An individual API (Application Programming Interface) key is generated for each admin interface user, which needs to be used to access the search interface. In practice the end user accesses the search interface with a URL in the following format:

```
http://[IP address]:[port]/renderer
?use=[index]
&login=[username]
&key=[API key]
&name=[renderer]
```

An admin user and a normal user were created for the admin interface. Privileges for querying the main index were given to the normal user. The normal user's API key was used to create a URL that the company personnel could use to access the search interface. With authentication enabled, a page with a login form as seen in Figure 8 was shown prior to accessing the search interface itself. After logging in, the end users are brought to the search interface seen in Figure 7.



Username or password is empty

Figure 8. OSS search interface login form.

4.3.8 Scheduler

Scheduled jobs, which can include various tasks, can be automatically executed at predetermined dates and/or times. These are defined with cron expressions. A cron expression is a string that consists of five to seven fields that together describe the schedule. A number of values or special characters can be used for each field. The fields and their allowed values and characters are listed in Table 9, and the special characters are explained in Table 10. (Terracotta, Inc. 2020)

Table 9. The fields of a cron expression and their allowed values and characters.

Name	Allowed values	Allowed special characters
Seconds	0-59	, - * /
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day of month	1-31	, - * / L W
Month	0-11 or JAN-DEC	, - * /
Day of week	1-7 or MON-SUN	, - * ? / L #
Year	Empty or 1970-2099	, - * /

Table 10. Explanations for the cron expression special characters.

Special character	Explanation
,	Used to specify multiple values
-	Used to specify a range of values
*	“All values”

/	Used to specify increments
?	“Any value”
L	“Last”
W	“Weekday”
#	“Nth weekday of the month”

Running a crawl indefinitely needlessly takes up system resources. Because of this, the crawler was initially run once manually, and a scheduled job was defined to run the crawl automatically. It includes a single task, which is starting the file crawler and running it once. The cron expression seen in Table 11 was used. The expression translates to “every year, every day of the week every month, every hour”, or in other words, “every hour”.

Table 11. Cron expression used to schedule automatic crawling.

Seconds	**Minutes**	**Hours**	**Day of month**	**Month**	**Day of week**	**Year**
0	0	*	?	*	*	*

The company supplied material representative of the real network drives’ content and structure for testing the designed OSS instance. The test instance successfully crawled the shares on the virtual machine and the search results were successfully opened from the host Windows. With this, the design was deemed ready for implementation in the real environment.

5 DEPLOYMENT

After a successful implementation in the test environment was demonstrated to the representatives, the deployment was done on-site within a four-day period. This section describes the deployment process, the challenges encountered during it and how they were solved.

5.1 The environment and scheduling

A new user had been created on the RHEL server for the sole purpose of handling the administration of OSS. The OSS archive was downloaded into the user's home directory and unpacked into its own folder. The OSS folder includes the server's files and scripts for starting and stopping the server. The scripts were modified to ensure that there is only one instance of the server running at any one time. This is done by forcefully stopping all Java processes in the stopping script, and running it before starting the server. The modified scripts can be found in Listing 3 and Listing 4, respectively.

Listing 3. Modified server starting script.

```
#!/bin/sh
/home/[ossuser]/opensearchserver/stop.sh

# Move to the directory containing this script
cd `dirname "$0"`
#
LANG=en_US.UTF-8
export LANG
JAVA_OPTS="$JAVA_OPTS -Dfile.encoding=UTF-8 -Djava.protocol.handler.pkgs=jcifs"

# The directory containing the indexes (must be exported)
OPENSEARCHSERVER_DATA=data
export OPENSEARCHSERVER_DATA

# The TCP port used by the server
SERVER_PORT=9090

# Any JAVA options. Often used to allocate more memory. Uncomment this line to al
locate 1GB.
JAVA_OPTS="$JAVA_OPTS -Xms6G -Xmx6G"

# Starting the server
eval java $JAVA_OPTS -jar opensearchserver.jar \
        -extractDirectory server \
        -httpPort ${SERVER_PORT} \
        -uriEncoding UTF-8 \
        >> "logs/oss.log" 2>&1 "&"

# Writing the PID
```

```
echo $! > "logs/oss.pid"
```

Listing 4. Modified server stopping script.

```
#!/bin/sh
# Move to the directory containing this script
cd `dirname "$0"`

# The location of the PID file
OSS_PID_FILE=logs/oss.pid
if ! [ -f "$OSS_PID_FILE" ]; then
    echo "PID file not found. Stop aborted."
    killall -9 java
    exit 1
fi

#Extract the PID
OSS_PID=`cat "$OSS_PID_FILE"`

# Check if the process exists
kill -0 $OSS_PID >/dev/null 2>&1
if [ $? -gt 0 ]; then
    echo "No matching process was found. Stop aborted."
    killall -9 java
    exit 1
fi

# Stopping the process and removing the PID file
kill $OSS_PID & rm $OSS_PID_FILE
killall -9 java
```

To reboot the OSS server every day at 6:30 o'clock, the following crontab was defined (Hess 2019):

```
30    06    *    *    *        /home/[ossuser]/ossreboot.sh
```

The crontab points to the reboot script in Listing 5 which runs the stop script, waits a few seconds and then runs the start script.

Listing 5. Server reboot script.

```
#!/bin/sh
sh /home/[ossuser]/opensearchserver/stop.sh
sleep 15
killall -9 java
sh /home/[ossuser]/opensearchserver/start.sh
```

To start and stop the OSS server with the server machine itself, init or RC (Run Command) scripts can be defined. These scripts run automatically at different states or *runlevels* of the system, depending on which RC folder they were added to. The first one, in Listing 6, runs the start script when the server starts, and was added into the folder */etc/rc3.d/*. The other script, in Listing 7, runs the stop script when the server is shutting down, and was added into the folder */etc/rc6.d/*. The naming of the scripts determines the parameters with which they are run (K for stop, S for start) and in which order they are run (01 first, 99 last). (Hussain 2013)

Listing 6. Automatic starting script *K01osserver.sh*.

```
#!/bin/sh
su [ossuser] -c "/home/[ossuser]/opensearchserver/start.sh"
```

Listing 7. Automatic stopping script *S99ossserver.sh*.

```
#!/bin/sh
su [ossuser] -c "/home/[ossuser]/opensearchserver/stop.sh"
```

5.2 OpenSearchServer setup

The OSS instance was for the most part set up as per the design presented in the previous section. Some challenges were encountered, however, and deviations from the original plan had to be made. These changes are examined in this section.

5.2.1 Analyzers

The *WinURLAnalyzer* was slightly modified. Whereas before the Samba file/folder paths were transformed into the following Windows format

```
file:\\mappedshare\share\folder\file
```

it was discovered that in place of the mapped name, the IP address of the server could be used instead. Additionally, it was discovered that the format didn't work in Mozilla Firefox, which was widely used in the company. Firefox requires five forward slashes in the beginning of the path, so the final format looked like the following:

```
    file:\\\\\192.168.1.1\share\folder\file
```

Modern browsers prevent opening local file links (even ones pointing to mapped network drives) by default for security reasons. This functionality can be enabled both on Chrome and Firefox by installing an extension. On Chrome the links could then be opened in the browser itself, while Firefox could open them in Windows Explorer or a default application. The links could also be copied and pasted into the address bar of Windows File Explorer, but this is quite inconvenient and presented another problem, which couldn't be solved. File paths that contain spaces or umlauts are displayed correctly in the search results, but will be mangled when copied to File Explorer.

5.2.2 Crawler

A single crawl location was defined for each share. The credentials of a user with access to all of the shares and all of their files were used to define the crawl locations. Several folders were manually excluded from the crawl at the discretion of one of the company's representatives.

When running the crawl for a single share, it was noticed that the crawl process would abruptly stop and leave most of the files unindexed. Upon rerunning the crawl and reviewing OSS's log files, it became apparent that a single folder containing several PowerPoint files was the cause of the issue. At first it seemed that the large size of some of the files resulted in the parser not being able to read their metadata, though the parser should have ignored the abnormally large files in the first place. Increasing the file size limit for the specific parser didn't solve the issue. Inspecting the file permissions, it seemed that the crawler should have had no problems accessing the files. In the end, the issue was resolved by changing the permissions that the crawlers extracted from *file & share permissions* to just *file permissions*.

5.2.3 Renderer

The company logo was transferred to the OSS images folder and was used in the header of the renderer. By default, the renderer included a viewer element for each search result that could be used to open the file. This feature didn't work reliably and was disabled. The file path *winDir* had to be URL decoded in order to display spaces and umlauts correctly, and a regex pattern was used to shorten it. A file path stored as

```
file:\\\\\192.168.1.1\share\folder\file
```

was displayed in the search results as

```
share\folder\file
```

The link points to the same file, but the displayed path is much more compact and readable. For an unknown reason, the first forward slash had to be omitted for this to work.

5.2.4 Authentication

With authentication otherwise set up as per the design, the credentials could be inserted into the credentials index. This turned out to be a non-trivial task. Since the end users should have had access to the same files that they would have on the Samba shares, the index should have included the same users as the RHEL server itself. There were dozens of users, and many of them belonged to multiple groups. No fully automated approach was found for extracting the users' credentials from the server.

To insert user credentials into the index manually, OSS's manual update feature was used. There are three ways to insert new documents into an index; either by using a form or uploading a text file in XML (Extensible Markup Language) or CSV (Comma-Separated Values) format. Using the form would've been impractical, since each of the values for the fields need to be inserted manually, and there is no way to add or change the value or values of a single field later. Using shell scripts, it is possible to export unix usernames and the groups they belong to. These could then be formatted into either XML or CSV. Parsing the CSV format turned out to be difficult, and XML was deemed to be the best option. A single user would be described in XML format as seen in Listing 8. The XML file describing one or more users would then be uploaded to OSS.

Listing 8. User credentials definition in XML.

```
<index>
    <document>
        <field name=”username”>
            <value>username</value>
        </field>
        <field name=”password”>
```

```
                <value>password</value>
            </field>
            <field name=”groups”>
                <value>group1</value>
            </field>
            <field name=”groups”>
                <value>group2</value>
            </field>
        </document>
</index>
```

The issue of password extraction still remained. In an ideal situation the user’s password would be exported from the server in an encrypted form and stored in the credentials index. This would be convenient for the end users since they would only need to memorize one password. In theory using the same password wouldn’t have been a problem on its own since an intruder wouldn’t have access to the files themselves even if they had access to the search interface. RHEL and OSS use different encryption algorithms for password encryption. As such, a password typed in by an end user in the search interface wouldn’t have matched the one exported from the server into the credentials index. Approaches to circumvent this issue were explored (such as overloading OSS’s encryption function) but none were successful.

In the end, the script seen in Listing 9 was used to export user credentials from the server. The script took a username and a password as inputs and output the user’s credentials into a text file in XML format. The password was stored in plain text in the file, and encrypted by the *CryptAnalyzer* during indexing. The password used for OSS needed to be different from the one used on the server, and the XML file needed to be deleted immediately after uploading it to OSS.

Listing 9. User credentials export script.

```
#!/bin/sh

read -p "Enter username: " username
read -s -p "Enter password: " password
echo ""
{
echo "<index>"
echo "<document>"

echo "<field name='username'>"
echo "<value>$username</value>"
echo "</field>"

echo "<field name='password'>"
echo "<value>$password</value>"
echo "</field>"

for j in $(groups $username | cut -d: -f2); do
        echo "<field name='groups'>"
        echo "<value>$j</value>"
        echo "</field>"
done

echo "<field name='groups'>"
echo "<value>everyone</value>"
echo "</field>"

echo "</document>"
echo "</index>"
} > ./userxml/$username.txt
```

One issue with authentication still remained: it was discovered that documents that should have been visible to everyone were not. This turned out to be a trivial problem: for these kinds of documents, OSS extracted the term “everyone” into the field *groupAllow*. In other words, only a group called “everyone” had permission to view the documents. Each user had to be added to this group. This was hardcoded into the user export script seen in Listing 9.

5.3 Using and maintaining OpenSearchServer

The search interface was designed to be as straightforward as possible for the end user. Short instructions on accessing and using it were written. The original instructions written in Finnish can be found in Appendix 1. A translated version follows:

“*The search engine OpenSearchServer has been deployed within the company, and can be accessed from* [link redacted]. *The same link can be found in the* [share name redacted] *network drive.*

[screenshot of the network drive redacted]

You can log into the search interface using your own credentials (for example [username redacted]*).*

You can search for files and folders by name and content. You can only see results that you have permissions to. All material within the share is not searchable, however. If you feel that the search doesn’t find something that it absolutely should, send feedback (email below).

The search results include links to the file and its folder. You can copy these links and paste them into Windows File Explorer's address bar (not all links work, for example ones with umlauts or spaces).

Browsers prevent opening local file links by default, but you can alternatively open the file or folder depending on your browser:

- *Chrome: add this extension* [link redacted]
- *Firefox: add this extension* [link redacted] *and download and install a small program according to its instructions (requires computer restart to work properly)*

Note that Chrome opens the file/folder in the browser, whereas Firefox opens it either with the default application or in File Explorer."

In addition, instructions for maintaining the OSS instance were written in English. These instructions can be found in Appendix 2.

6 EVALUATION

As per the design research process, once the artefact i.e. search engine was deployed, it had to be evaluated. While a search engine can be evaluated with various performance metrics, it was decided that the performance and utility of the search interface would be evaluated by surveying the end users (Croft et al. 2015). As such, a survey for gathering quantitative and qualitative data from the users was planned. This section describes the survey, the analysis of its results and the improvements made to the implementation based on the results.

6.1 Survey and analysis

13 employees from different departments were selected by a company representative. The instructions on the usage of OSS were sent to them, and they were given two weeks to test the search interface. After the two-week period, an anonymous survey was sent to the them, and a week's time was given to respond. Open feedback was also accepted via e-mail. Seven of the 13 testers responded to the survey, and three individuals sent feedback via e-mail.

The survey's questionnaire consisted of five sections and 30 questions in total. Open feedback was accepted at the end of each section. A scale of one to five was used for 23 of the 27 closed questions. The answers to these 23 questions are summarized from Table 12 to Table 16. For most of these questions, a score of 1 signified the most negative and a score of 5 signified the most positive response. A score of 3 was reserved for a neutral, unsure or middle ground response. Other answers and feedback are analyzed for each section.

Instructions

Table 12. Questions on the instructions and their results.

Question	Lowest (1)	Average	Highest (5)
I understood how I could access the search engine.	Not at all	4.3	Fully
I understood what I could search and how I could search.	Not at all	4.1	Fully

I understood how I could open the document or folder from the search results.	Not at all	4.0	Fully
Installing the extension was	Difficult	3.5	Easy
The instructions were	Incomprehensible	3.6	Clear
The instructions were	Too long	3.3	Too short

Open feedback on the instructions:
"*Clear instructions, installing the extension was a little troublesome but this was more likely due to hardware issues.*"
"*Installing the extension itself was easy, but at least for Mozilla Firefox another file had to be downloaded before the results could be opened."*

The instructions were deemed mostly appropriate and understandable. All of the respondents used Mozilla Firefox as their browser of choice, and 6 of the 7 respondents installed the extension as per the instructions. Challenges with the extension were encountered by some respondents, but overall installing it was deemed easy enough.

Logging in

Table 13. Questions on logging in and their results.

Question	**Lowest (1)**	**Average**	**Highest (5)**
I could access the search interface (even remotely)	Never	3.7	Always
I understood which credentials I had to use to log in	Not at all	4.0	Fully
There were problems with logging in	Constantly	4.0	Never
Logging in was	Irritating	4.9	Effortless

Open feedback on logging in:
"*After installing the extension using the search interface was straightforward.*"
"*If the search interface wasn't used for a while, the user was automatically logged out. It would be better if it stayed logged in.*"
"*At the beginning of the test period there were constant problems with logging in, though later these problems disappeared.*"

For the most part accessing and logging into the search interface worked well enough. Some unknown causes resulted in the interface not being available for some respondents, though these issues were resolved on their own. Some respondents worked remotely and had to use a VPN (Virtual Private Network) to connect to the company's private network. Most respondents reported that their remote connection worked as expected. Only one respondent claimed they couldn't access the search interface at all, but they also paradoxically said that they had no problems logging in.

There were two ways to initially access the search interface: using the link in the instructions or using the link in the network drive. Three respondents said that creating a bookmark was the most convenient way to access the interface, while three others said that the link in the instructions was the most convenient. Only one respondent felt the link in the network drive was the most convenient.

Searching

Table 14. Questions on the search functionality and their results.

Question	**Lowest (1)**	**Average**	**Highest (5)**
Searching was	Slow	4.3	Fast
Suggestions in the search bar were	Useless	3.6	Useful
Different search terms worked	Poorly	3.7	Well
The search results were	Irrelevant	3.6	Appropriate

The search found what I expected it to find	Never	3.6	Always
The number of search results on one page was	Too large	3.0	Too small
The snippet of content in the search results was	Useless	4.3	Useful
I managed to open files or folders from the search results	Not at all	3.6	Easily
Opening a file or folder from the search results was	Useless	4.1	Useful
Filtering or sorting the search results was	Useless	4.0	Useful

Open feedback on searching:
"*There were duplicate file extensions in the filters, for example .DWG and .dwg. Could these be combined?*"
"*Searching was easy, could have been even faster.*"
"*Searching with a partial word should work, and the same file extensions regardless of case should be combined.*"
"*You had to remember the filename very closely, for example if you forgot an underscore, the search couldn't find the file.*"

Most respondents said that searching was fast. Suggestions in the search bar weren't entirely useful. Not all search terms worked as anticipated, though the results were mostly expected and relevant. Some expected individual files or folders couldn't be found, but later it was determined that these folders were excluded altogether. The amount of search results per page was appropriate, and the content snippets were deemed highly useful. The links were more often used to open files or folders. This feature was seen as very useful but didn't always work. The filtering and sorting features were mostly appreciated. One of the respondents said they wanted to sort the search results by date, which was already possible.

Utility

Table 15. Questions on the utility of the search engine and their results.

Question	Lowest (1)	Average	Highest (5)
I felt the search interface was useful	Not at all	4.0	Very much
I could use the (improved) search interface regularly in my work	No way	4.3	Surely

Open feedback on improving the search interface:
"*Searching with just the start of the word should be more accurate.*"
"*Searching with a partial filename would be good.*"
"*If possible, the search interface could be accessible with something else besides a browser.*"
"*Searching with a partial word should work, and the same file extension should be combined in the filters.*"
"*During the short test period I couldn't find any obvious improvements to be made.*"
"*I couldn't open the search interface directly from the link, this should be fixed.*"
"*The search could be more accurate.*"

The utility of the search interface was perceived as quite high. Most respondents felt that they could use it regularly, provided that the suggested improvements were implemented.

Appearance

Table 16. A question on the appearance of the search interface and its result.

Question	Lowest (1)	Average	Highest (5)
I liked the appearance of the search interface	Not at all	3.4	A lot

Open feedback on the appearance of the interface was sparse, with just one respondent raising clarity as an important aspect. The appearance was received neutrally for the most part, which can perhaps be interpreted as a success.

Other feedback received via e-mail included several testers reiterating other points raised in the survey. Many wished there was a way to search with any partial search term. One tester stated that the search engine reduced the amount of time spent on redundant browsing, though this required careful storage and naming of documents on the server.

6.2 Improvements

Based on the feedback received from the survey and via e-mail, some improvements to be made were determined:

- Enabling searching with any partial search term
- Improving relevancy with query template settings
- Combining similar extensions in the filter list
- Communicating clearly which drives, folders and files are included in the search

Initially, an edge n-gram filter was added into the *TextAnalyzer*. As it turned out, this only allowed the users to search with partial search terms that started at the beginning of a word. To rectify this, the filter was replaced with an n-gram filter. The difference is illustrated in the following:

```
search → sea, sear, searc, search
search → sea, ear, arc, rch, sear, earc, arch, searc, earch,
search
```

To try and improve relevancy of search results, the schema and the query template were modified slightly. A new field named *full* was added to the schema as a copy of the fields *title*, *content*, *fileName* and *url*. It was set to be processed by the *TextAnalyzer* and was added to the searched fields in the query template. In the template settings, the default operator OR was coupled with a mirror AND filter. What this effectively meant was that the search result scores were based on the OR operator, but the results were filtered with the AND operator, combining the “best of both worlds” (OpenSearchServer, Inc. 2020c).

Initially, no analyzer had been defined for the *fileExtension* field. This resulted in duplicate file extensions, only with different capitalization, in the filter list. This problem was solved by defining an *ExtensionAnalyzer* for the field, which transformed the file extensions into lower case.

Due to the changes made, the size of the index was expected to rise. Whereas before the disk space taken up by the index of over 40 thousand documents amounted to around 1.3 gigabytes, afterwards this climbed up to 3.3 gigabytes, which was still satisfactory.

The end users should be kept informed about which folders and files are included in the search system and which are not. Folders and files can be included or excluded arbitrarily at any point, but it is unlikely that this will happen often. No elegant solution to this was found, but one straightforward one could be to have a document listing the exclusions in the root of the network shares where the link to the search interface is located as well.

Unrelated to the survey itself, it was discovered that after the OSS server was restarted automatically and the automatic crawl job was executed, the file crawler kept running indefinitely. To prevent the crawl process from constantly taking up system resources, a scheduled job was defined that stopped the file crawler shortly after the server was restarted. In addition, running the file crawl every hour was deemed excessive. The crawl was instead scheduled to start one hour after the server restarted each day. This level of timeliness was acceptable, since not too many documents are added to the network drives each day and it is unlikely that the newest files would be the ones that the users search for.

After implementing the improvements based on feedback, the instructions on the usage of the search interface were ready to be distributed to the rest of the personnel. Further testing and feedback would be required to determine whether the made changes were effective and if any additional ones would be needed, but the implementation was considered to be ready for real use.

7 RESULTS, DISCUSSION AND CONCLUSIONS

This section lists the results of the research, and reflects upon both the results and the research process itself. Possibilities for further development and research are considered, and the conclusions of the research are presented.

7.1 Results

As a result of the research, the company's development needs related to information retrieval were identified. It turned out that these needs were not only related to information retrieval, but also to the underlying activities of knowledge management, information management and document management. The tacit and explicit knowledge possessed by some employees was not made available to others, and the lack of defined workflows and guidelines made sharing and retrieving information inefficient. In addition, having no document management or a robust search system in place resulted in employees struggling to find documents, and the N-programs presented problems of their own.

As another result of the research, alternative solutions for fulfilling the company's needs related to information retrieval were identified. These included two types of information technology artefacts: document management systems and information retrieval systems. Though an existing DMS is in use at the company, it is not utilized for the internal documents on the network drives. While the application of these systems was mostly considered with the improvement of information retrieval in mind, a DMS could have presented a more holistic solution and addressed some of the underlying issues. Though a DMS was not utilized to improve document management on the network drives as part of this research, the company may be interested in doing so in the future.

As the practical result of the research, an implementation of an existing open source search engine instance was designed and deployed in the company intranet. The first iteration of the implementation was evaluated and improvements were made based on the feedback. Overall the search engine was received well by the test users and was seen as potentially useful. After making the necessary adjustments, the instructions on the usage of the search engine were sent to the rest of the company personnel. Additional improvements may be implemented based on further testing and feedback.

7.2 Discussion

At the beginning of the research a search engine was already suggested as a candidate solution for addressing the company's information retrieval challenges. While this may have been justified, it may have narrowed the focus of the research too much. Though other alternatives were explored, a strong preference for implementing a search engine remained throughout the research process. Significant contributing factors to this were the practical delimitations of the research, as it was assumed early on that implementing a more demanding alternative, such as a document management system, would not have been feasible. One factor that compensated the limitations caused by remote work was the test environment, which enabled the design of the artefact to approximate the final implementation very closely.

Search engines are readily available but finding promising open source candidates and testing them takes time. While commercial alternatives may have proved to be more capable, they had to be excluded due to cost restrictions. The open source alternatives were often dated and had poor documentation, which made testing even more difficult. Significant hindrances were encountered with many of the tested alternatives, and determining whether to try and solve these issues or move on to another candidate was challenging. With OpenSearchServer, the tradeoffs between the ease of implementation and level of sophistication and control were deemed acceptable.

Though there are numerous metrics that may have been used to evaluate the artefact, it was decided that user feedback would be the most valuable and efficient form of evaluation in order to quickly improve the implementation. Though the results from the initial evaluation were positive and suggest that the search engine may be effective, the success of the made changes can only be confirmed with further testing and user feedback. The performance of the engine and especially the relevance of the search results may be improved further by utilizing some of the features that were not discussed in this thesis, such as the learner and classifier components.

Maintaining the search engine may prove challenging due to the lack of documentation, though the implementation itself and the supplied instructions were designed to mitigate this as much as possible. Another shortcoming is that no safeguards or safety measures were considered in case of hardware or software failures, though no significant damages should be suffered by the company even if the search engine was disabled for whatever reason. Currently the search engine's practical worth to the company is unknown, but in the future the engine's reliability may be improved, for example with the index replication feature.

Even in the case that the search engine proves itself useful, it may only be a temporary solution. A search engine doesn't solve the underlying issues related to knowledge, information and document management. In the future, the company may be interested in at least implementing a document management system, and perhaps also planning and executing enhancements to the other management activities as well.

7.3 Conclusions

The goal of this thesis was to improve the company's information retrieval. The company's development needs were identified and a literature review into alternative solutions was conducted. It was determined that an information retrieval system, or a search engine, was a more viable option to implement. A test environment was setup and alternative search engines were explored, compared and tested, and one was chosen for implementation. The chosen search engine instance was designed and deployed at the company, and preliminary evaluations suggested that it increased the efficiency of information retrieval from the server's network drives, and that its utility for the company was potentially high. Room for enhancement remains, and utilizing a document management system for the network drives and improving both knowledge management and information management activities were suggested as possible future developments for the company.

REFERENCES

Adam, A. (2007). *Implementing Electronic Document and Record Management Systems*. Auerbach Publications. [online]. Available from: https://www.taylorfrancis.com/books/9780849380600 [Accessed May 27, 2020].

Ambar LLC. (2020). Ambar - Document Search Engine · An open-source document search engine with automated crawling, OCR, tagging and instant full-text search. [online]. Available from: https:/ambar.cloud/ [Accessed June 2, 2020].

Anon. (2020a). Samba - opening windows to a wider world. [online]. Available from: https://www.samba.org/ [Accessed June 6, 2020].

Anon. (2020b). Setting up Samba as a Standalone Server - SambaWiki. [online]. Available from: https://wiki.samba.org/index.php/Setting_up_Samba_as_a_Standalone_Server [Accessed June 3, 2020].

Apache Solr Software Foundation. (2020). Apache Solr -. [online]. Available from: https://lucene.apache.org/solr/ [Accessed June 3, 2020].

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. New York: ACM press.

Bancilhon, L. (1999). Steps involved prior to the implementation of an intranet search engine in a Web-based intranet environment. *SA Journal of Information Management*, 1(1). [online]. Available from: https://sajim.co.za/index.php/sajim/article/view/67 [Accessed June 2, 2020].

Bootstrap Core Team. (2020). Bootstrap · The most popular HTML, CSS, and JS library in the world. [online]. Available from: https://getbootstrap.com/ [Accessed June 10, 2020].

van Brakel, P. (2003). In need of document management competencies. *South African Journal of Information Management*, 5(4). [online]. Available from: https://www.researchgate.net/publication/272644087_In_need_of_document_management_competencies [Accessed May 30, 2020].

Büttcher, S., Clarke, C.L.A. and Cormack, G.V. (2010). *Information Retrieval - Implementing and Evaluating Search Engines*. [online]. Available from: https://pdfs.semanticscholar.org/9d64/eaf01183ccbf3f2921a00ba3388c817bd72b.pdf?_ga=2.18533177.188150087.1581768184-337340091.1579695328 [Accessed February 15, 2020].

Carpenter, D. (2020). voidtools. [online]. Available from: https://www.voidtools.com/ [Accessed June 3, 2020].

Chen, X.H., Snyman, M. and Sewdass, N. (2005). Interrelationship between document management, information management and knowledge management. *South African Journal of Information Management*, 7(3). [online]. Available from: https://www.researchgate.net/publication/228617437_Interrelationship_between_document _management_information_management_and_knowledge_management [Accessed May 19, 2020].

Cho, J. and Garcia-Molina, H. (2002). Parallel Crawlers. In *Proceedings of the 11th international conference on World Wide Web*. [online]. Available from: https://oak.cs.ucla.edu/~cho/papers/cho-parallel.pdf [Accessed February 15, 2020].

Christen, M. (2020). Home - YaCy. [online]. Available from: https://yacy.net/ [Accessed June 3, 2020].

Cleveland, G. (1995). *Overview of document management technology*. IFLA, Universal dataflow and telecommunications core programme.

Cole, B. (2005). Search Engines Tackle the Desktop. *Computer*, 38(3), pp.14–17. [online]. Available from: http://www.dbnet.ece.ntua.gr/~dalamag/pub/r3014.pdf [Accessed February 15, 2020].

Croft, W.B., Metzler, D. and Strohman, T. (2015). *Search Engines - Information Retrieval in Practice*. Addison-Wesley Reading. [online]. Available from: https://www.semanticscholar.org/paper/Search-Engines-Information-Retrieval-in-Practice-Croft-Metzler/c029baf196f33050ceea9ecbf90f054fd5654277 [Accessed February 8, 2020].

Docile, E. (2019). How to install and configure samba on RHEL 8 / CentOS 8 - LinuxConfig.org. [online]. Available from: https://linuxconfig.org/install-samba-on-redhat-8 [Accessed June 3, 2020].

Docker Inc. (2020). Empowering App Development for Developers | Docker. [online]. Available from: https://www.docker.com/ [Accessed June 4, 2020].

Elasticsearch B.V. (2020). Open Source Search: The Creators of Elasticsearch, ELK Stack & Kibana | Elastic. [online]. Available from: https://www.elastic.co/ [Accessed June 3, 2020].

France Labs. (2020). Datafari Enterprise Search. [online]. Available from: https://www.datafari.com/en/ [Accessed June 3, 2020].

Free Software Foundation, Inc. (2020). gnu.org. [online]. Available from: https://www.gnu.org/licenses/gpl-3.0.html [Accessed June 4, 2020].

Göker, A. and Davies, J. (2009). *Information retrieval: searching in the 21st century*. John Wiley & Sons. [online]. Available from: https://ia600300.us.archive.org/0/items/IrSearchingInThe21stCentury/0470027622_Information.pdf [Accessed February 15, 2020].

Grehan, M. (2002). How Search Engines Work. In *Search Engine Marketing: The Essential Best Practice Guide*. [online]. Available from: https://www.searchenginewatch.com/wp-content/uploads/sites/25/2016/01/how-search-engines-work-mike-grehan.pdf [Accessed February 15, 2020].

Hernad, J.M.C. and Gaya, C.G. (2013). Methodology for Implementing Document Management Systems to Support ISO 9001:2008 Quality Management Systems. *Procedia Engineering*, 63, pp.29–35. [online]. Available from: https://linkinghub.elsevier.com/retrieve/pii/S1877705813014380 [Accessed May 19, 2020].

Hess, K. (2019). Automate your Linux system tasks with cron. *Enable Sysadmin*. [online]. Available from: https://www.redhat.com/sysadmin/automate-linux-tasks-cron [Accessed June 5, 2020].

Hevner, R. et al. (2004). Design science in information systems research. *MIS Quarterly*, 28(1), pp.75–105. [online]. Available from: https://sites.google.com/site/yamilejaime/DESIGNSCIENCEININFORMATION.pdf [Accessed June 10, 2020].

Hussain, S. (2013). How To Configure a Linux Service to Start Automatically After a Crash or Reboot – Part 2: Reference. *DigitalOcean*. [online]. Available from: https://www.digitalocean.com/community/tutorials/how-to-configure-a-linux-service-to-start-automatically-after-a-crash-or-reboot-part-2-reference [Accessed June 7, 2020].

Kowalski, G. (2011). *Information Retrieval Architecture and Algorithms*. Springer Science & Business Media.

Mandalka, M. (2020). Open Semantic Search: Your own search engine for documents, images, tables, files, intranet & news. [online]. Available from: https://www.opensemanticsearch.org/ [Accessed June 3, 2020].

Manning, C., Raghavan, P. and Schuetze, H. (2009). *Introduction to Information Retrieval*. Cambridge university press. [online]. Available from: https://ds.echhost.com/jspui/bitstream/123456789/2452/1/00776216.pdf [Accessed February 13, 2020].

Mutai, J. (2019). How to Disable SELinux on RHEL 8 / CentOS 8. *ComputingForGeeks*. [online]. Available from: https://computingforgeeks.com/how-to-disable-selinux-on-rhel-8-centos-8/ [Accessed June 3, 2020].

OpenSearchServer, Inc. (2020a). OpenSearchServer | Open Source Search Engine and Search API. [online]. Available from: https://www.opensearchserver.com/ [Accessed June 3, 2020].

OpenSearchServer, Inc. (2020b). OpenSearchServer Documentation - Discovering the main concepts. [online]. Available from: https://www.opensearchserver.com/documentation/tutorials/functionalities.md [Accessed June 4, 2020].

OpenSearchServer, Inc. (2020c). OpenSearchServer Documentation - Improving relevancy with ‘Mirror AND filter’. [online]. Available from: https://www.opensearchserver.com/documentation/faq/querying/improving_relevancy_with_mirrorandfilter.md [Accessed June 4, 2020].

OpenSearchServer, Inc. (2020d). OpenSearchServer Documentation - Linux (generic). [online]. Available from: https://www.opensearchserver.com/documentation/installation/linux.md [Accessed June 4, 2020].

Porter, M.F. (2006). An algorithm for suffix stripping. *Program*, 40(3), pp.211–218. [online]. Available from: https://www.emerald.com/insight/content/doi/10.1108/00330330610681286/full/html [Accessed February 15, 2020].

Red Hat Inc. (2020). Red Hat software downloads for developers. *Red Hat Developer*. [online]. Available from: https://developers.redhat.com/products/ [Accessed June 4, 2020].

van Rijsbergen, C.J. (1979). *Information retrieval*. [online]. Available from: http://www.dcs.gla.ac.uk/Keith/Preface.html [Accessed February 15, 2020].

Saracevic, T. (1999). Information science. *Journal of the American Society of Information Science*, 50(12), pp.1051–1063. [online]. Available from: https://www.scribd.com/document/267953508/87-Saracevic-Information-Science [Accessed February 15, 2020].

Sathiadas, J.P. and Wikramanayake, G.N. (2003). Document management techniques and technologies. In *Proceedings of the 5th international information technology conference*. pp. 40–48. [online]. Available from: https://icter.org/conference/icter2016/sites/default/files/icter/IITC2003book.pdf#page=46 [Accessed May 29, 2020].

SearchBlox Software, Inc. (2020). SearchBlox AI-Driven Search. *SearchBlox AI-Driven Search*. [online]. Available from: http://www.searchblox.com/ [Accessed June 3, 2020].

Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.*, 24(4), pp.35–43. [online]. Available from: http://sifaka.cs.uiuc.edu/course/410s12/mir.pdf [Accessed February 15, 2020].

Terracotta, Inc. (2020). Cron Trigger Tutorial. [online]. Available from: http://www.quartz-scheduler.org/documentation/quartz-2.3.0/tutorials/crontrigger.html [Accessed June 7, 2020].

VMWare, Inc. (2020). Download VMware Workstation Player | VMware. [online]. Available from: https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html [Accessed June 4, 2020].

Witten, I.H., Moffat, A. and Bell, T.C. (1999). *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann. [online]. Available from: http://cyber.sibsutis.ru:82/%D0%A1%D0%9F%D0%98/%D0%9F%D0%B5%D1%80%D0%B2%D0%B0%D1%8F%20%D1%87%D0%B0%D1%81%D1%82%D1%8C/Managing%20Gigabytes;%20Witten,%20%D0%9Coffat,%20Bell.pdf [Accessed February 15, 2020].

Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, 38(2), pp.6-es. [online]. Available from: http://portal.acm.org/citation.cfm?doid=1132956.1132959 [Accessed February 15, 2020].

APPENDIX 1. Finnish instructions on the usage of OpenSearchServer

OpenSearchServer-hakumoottorin käyttöohjeet

Meillä on nyt otettu käyttöön hakumoottori OpenSearchServer, johon pääset tästä linkistä [hyperlink redacted]. Sama linkki löytyy myös pikakuvakkeena [network share redacted]-verkkokansiosta:

[screenshot redacted]

Voit kirjautua hakuun oman koneesi tunnuksilla (esim. [username redacted]).

Voit hakea tiedostoja ja kansioita niiden nimen ja sisällön perusteella. Näet vain sellaiset tulokset, joihin sinulla on pääsyoikeudet. Kuitenkaan aivan kaikki materiaali jaoissa ei ole haettavissa. Jos koet ettei haku löydä jotain mitä ehdottomasti pitäisi, anna palautetta (sähköpostiosoite alempana).

Hakutuloksissa on linkit tiedostoon ja sen kansioon. Voit kopioida linkin Windowsin resurssienhallinnan osoitepalkkiin (kaikki linkit eivät toimi kopioidessa, esim. ääkköset tai välilyönnit).

Selaimet estävät tavallisesti linkkien avaamisen suoraan, mutta vaihtoehtoisesti pääset avaamaan tiedoston tai kansion selaimestasi riippuen:

- Chrome: lisää tämä laajennus [hyperlink redacted] selaimeen
- Firefox: lisää tämä laajennus [hyperlink redacted] selaimeen, ja asenna sen ohjeiden mukaan pieni ohjelma koneellesi (vaatii koneen **uudelleenkäynnistyksen** toimiakseen kunnolla)

Huomaa, että Chrome avaa tiedoston/kansion selaimessa, kun taas Firefox oletusohjelmalla/resurssienhallinnassa.

APPENDIX 2. Instructions on the maintenance of OpenSearchServer

OpenSearchServer search engine – admin instructions

This document instructs in the administrative use of the OpenSearchServer (OSS) search engine. It assumes some familiarity with using the command line interface, Linux, and computers in general. All of the important functions are covered here, but for more information you can refer to the official documentation.

The most important tasks the reader should be able to accomplish are:

- Starting/stopping the server (Section 1).
- Adding/deleting/modifying users (Sections 6 and 12).
- Adding/deleting documents (Section 8).

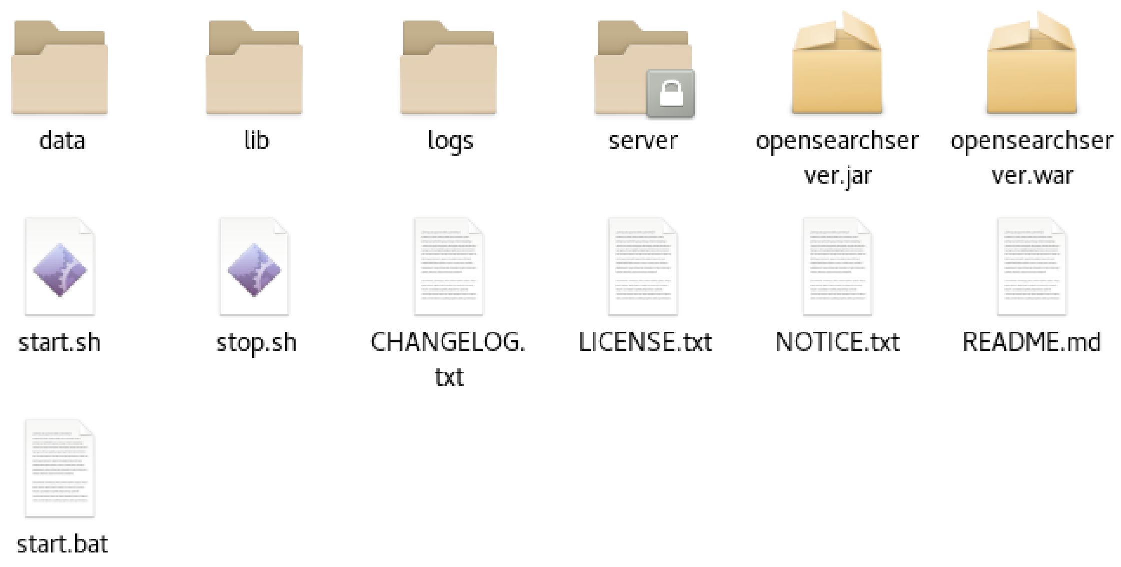
(continues)

Contents

(continues)

1. Server files, starting and stopping

The OSS folder can be accessed by logging in to the company server (using Putty, for example) as the user *[username redacted]*. The OSS folder is in the home folder of the user, and contains the following:



Usually, modifying any of the files is unnecessary.

In short, the OSS server can be started and stopped with the *start.sh* and *stop.sh* script files. It will run on port 9090 by default and its web interface can be accessed with any browser connected to the private local network by typing the IP and port into the address bar (for example 192.168.160.6:9090). You can find the IP address of the server with the *ifconfig* command, for example.

The OSS server has been configured to automatically start and stop along with the server machine itself, and it restarts every morning at 06:30. The *start.sh* script will also first run the *stop.sh* script so that only a single instance of OSS is running at any time, and *stop.sh* will forcefully stop any Java processes, effectively ensuring that OSS shuts down.

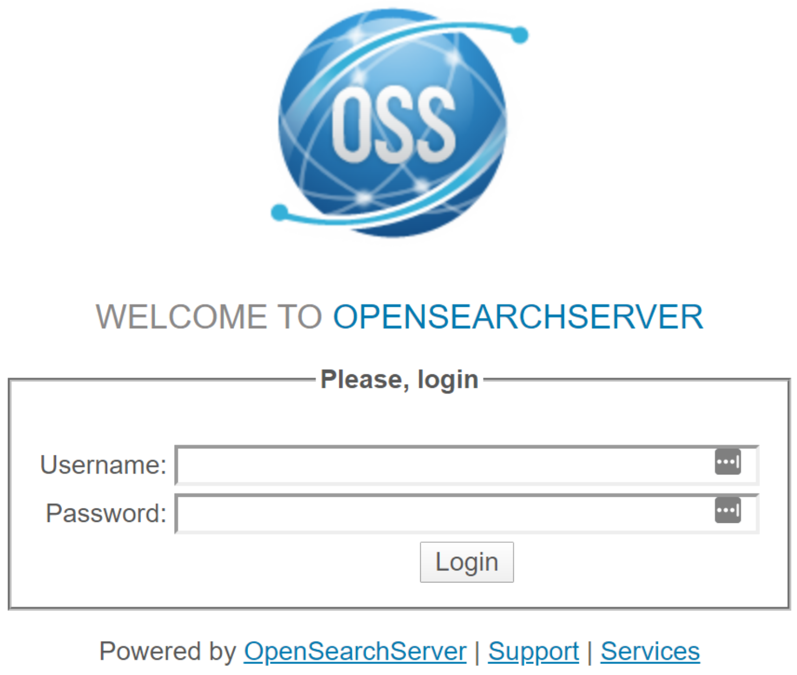
The OSS server has been configured to use a maximum of 6 gigabytes of memory. This can be edited in the *start.sh* file (JAVA_OPTS="$JAVA_OPTS -Xms6G -Xmx6G").

(continues)

APPENDIX 2. (continued)

2. Logging in and web interface

After typing in the address, you should be greeted with the following login screen:



WELCOME TO OPENSEARCHSERVER

Please, login

Username:

Password:

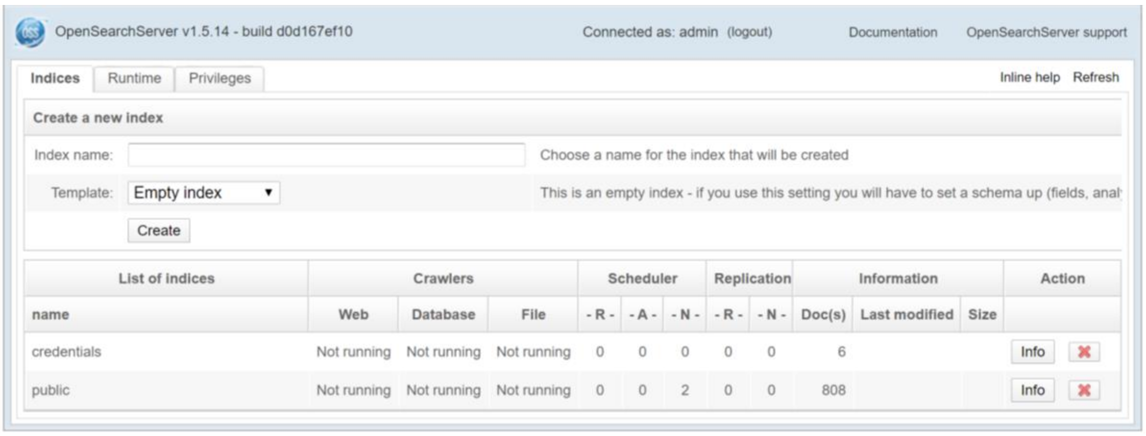
Login

Powered by OpenSearchServer | Support | Services

Only two users are available for logging in to the **admin interface**: *admin* and *haku*. The latter is only used to provide access to the **search interface** for the **end users**, and there is usually no reason to log in as *haku* here.

Note that these users are ***not*** the same as the ones used for logging in to the search interface itself. This distinction will be made clearer later.

After logging in as admin, you will see the following:



OpenSearchServer v1.5.14 - build d0d167ef10   Connected as: admin (logout)   Documentation   OpenSearchServer support

Indices | Runtime | Privileges   Inline help   Refresh

Create a new index

Index name: Choose a name for the index that will be created

Template: Empty index ▾   This is an empty index - if you use this setting you will have to set a schema up (fields, anal

Create

List of indices	Crawlers			Scheduler			Replication		Information			Action
name	Web	Database	File	- R -	- A -	- N -	- R -	- N -	Doc(s)	Last modified	Size	
credentials	Not running	Not running	Not running	0	0	0	0	0	6			Info ✖
public	Not running	Not running	Not running	0	0	2	0	0	808			Info ✖

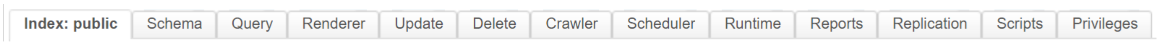
(continues)

Tip: since the interface may not always automatically refresh, you can do this manually from the top right corner of the window.

At this point only three tabs are available:

- Indices
    - List of indices. An index is essentially a database that enables fast searching of files.
    - Two indices are used, one for the **documents** themselves and one for the **credentials** of the users that can log in to the search interface. No new indices should need to be created.
- Runtime
    - Information about the system and its resources, as well as logs and advanced features. Can be helpful for troubleshooting.
- Privileges
    - Here users such as *admin* and *haku* and their privileges can be added. Usually there is no need to create new users here.

After selecting an index by clicking its row in the list, more tabs will appear:



Most of these will be discussed in detail in the following chapters.

- **Schema**
    - Defines what information is stored for each document in the index and how the information is processed or analyzed beforehand.
- **Query**
    - Defines templates for querying and returning information from the index.
- **Renderer**
    - Defines the search interface for the end users.
- **Update**
    - Can be used to manually add or update documents in the index. Not necessary for the **document** index but will be used for the **credentials** index.
- **Delete**
    - Can be used to manually delete documents in the index. Again, only used for the credentials index.
- **Crawler**
    - Defines the crawlers that search and retrieve files for indexing.
- **Scheduler**
    - Defines jobs that are executed automatically at specified dates and times.
- **Reports**
    - Can be used to generate reports.

(continues)

- **Replication**
  - Can be used to replicate indices.
- **Scripts**
  - Can be used to create scripts.

3. Schema

This tab defines the information stored for each document in the index and how that information is retrieved and processed beforehand. It contains the following subtabs:

Fields

Fields are the individual pieces of information stored for each document. These include the filename, the file type, the file path, the access rights, and others. To individualize each document, a unique field is needed. The URL (essentially the file path) is a good choice.

Most of the fields are self-explanatory, but a couple can be a bit vague.

- Title
  - Metadata title of the document (not necessarily the same as the filename)
- Autocomplete
  - Used to provide autocompletion/suggestions in the search bar based on the filename and the file contents.
- WinDir and WinURL
  - The folder and file paths in Windows format, respectively.
- Share
  - The share name (for example *public*)

The field settings determine the following:

- Indexed
  - Content of the field is indexed, and queries can be executed within it.
- Stored
  - Content of the field is stored as is, i.e. before indexing/processing/analyzing.
  - Can be optionally compressed.
  - Useful for displaying the exact filenames and snippets of contents in the search results.
- Term vector
  - Indicates that a field is a list of items.
  - Useful for storing multiple values, such as access rights.
  - Needed for generating snippets (offsets are needed as well).
- Analyzer
  - Used to process the content of the field.

(continues)

- Copy of
    - The initial value for this field is copied from the initial value of some other field or fields.

Analyzers

Analyzers are (optionally) used to process the contents of the fields. For example, the title, filename and content fields are processed by the TextAnalyzer to make storing and querying them more efficient. Analyzing can be done during both indexing and querying. Some analyzers are included by default, but a few have been created or modified.

An analyzer can be defined for each language separately, and it consists of **tokenizers** and **filters**:

- Tokenizing
    - Individual pieces of text or characters are identified and separated into so-called tokens.
    - Unwanted characters, such as whitespaces, can be discarded.
    - Mainly two tokenizers are used: the standard and the keyword tokenizers.
        - The former is used for traditional text (“this text” becomes “this” and “text”).
        - The latter is used to process the entire contents of the field as a single token (a file path, for example).

- Filtering
    - After tokenizing, the tokens can be filtered. Filtering can help in decreasing the size of the index and improving the relevance of the search results.
    - Some examples include:
        - Recognizing certain patterns and discarding or replacing those tokens.
        - Removing stop words (common words such as *the*).
        - Removing characters such as *Ä* and *Ö*.
        - Removing prefixes and suffixes such as *-lla* or *-sta*.
        - A significant part of indexing is stemming, which truncates words into their common form. For example, the words *kalastus* and *kalastaa* could both be stemmed into their common form, *kala*. Filters for stemming words in specific languages are available.

In addition to decreasing the size of the index and improving query performance, the analyzers also help format the search results. The output of an analyzer can be tested while editing.

Parser list

The parsers process the files found by the crawlers and extract the various fields for indexing. You do not have to worry about how they work, but it may be desirable to increase the file size limit which is typically around 30 Mb. This limit can be increased on a per parser basis.

(continues)

Stop words

Here lists of stop words (common words) for each language can be added. The words are separated by newlines and there should not be an empty line at the end of the list. Stop words can be used with a stop filter in an analyzer. This decreases the size of the index.

Autocompletion

Here the field used for autocompletion can be defined and tested. The default settings should be fine.

Authentication

To implement authentication, information about each document's access rights needs to be stored in some fields. These fields can be defined here. The rights can be stored in the same index as the rest of the fields, or in a separate index. If no rights are found for the document, a user or a group could be given access by default.

4. Query

This tab defines the information that is searched for and returned when the end user types in a search query. As for the general settings, most of the defaults should work fine, but the default operator OR could be a better choice in order to return relevant results.

Searched fields

These are the fields that a query is executed within. The user should be able to search for files based on their title, name, content, and location. The title and filename are given a larger weight, and for each field inexact queries are allowed with the phrase slop setting.

The mode setting has four options:

- Pattern
  - No processing done to keywords: for example, "this text" is a single keyword.
- Term
  - Special characters are removed, and each keyword is queried separately.
- Phrase
  - Special characters are removed, and the whole phrase is queried.
- Term & phrase
  - As above, but both the terms and the whole phrase are queried.

(continues)

Returned fields

These are the fields that will be available to display in the search results.

Faceted fields

These are the fields that will be used for filtering. The date filter is available by default and is not listed here.

Snippet fields

These are the fields that are used for snippets.

Sorted fields

Here some default sorts could be defined. Ordering by relevance is usually desired, so defining any sorts here should be unnecessary. The user will be able to sort the results in different ways, but these are defined in the *Renderer* tab.

Filters

The name of this tab can be a bit misleading. Setting a filter here filters all results and the end user has no control over it, so only the mirror AND filter is added. Coupled with the default operator OR in the general setting, this setting should improve the relevance of the search results.

5. Renderer

The renderer defines the search interface. From the list of renderers, selecting *view* opens up the search interface, which is also visible to the end users. After you open the interface, note that the address bar shows which user the interface is being accessed with (for example *&login=admin*). For the end users, this would be *haku* instead. **Make sure not to share the admin address with anyone who should not access the admin interface!**

Selecting *edit* reveals new tabs for editing the renderer. The general settings are used for labels and appearance.

Fields

These are the pieces of information that will be displayed for each result.

(continues)

Filters

These are the filters that the end user can select to filter out results.

Sorts

These are the manual sorts the end user can select to reorder the results. The hyphen (-) sets the order from highest to lowest.

CSS style

CSS is used to style web pages. In the simplest form, it can be used to change the color and size of text, as well as positions of the elements that the page consists of. Most of the styles used are the default ones. Styles with a dot are for classes (for example *.osscmnrdr*), and styles with a hashtag are for ids (for example *#ossautocomplete*). Fields can be given specific classes in the *Fields* tab.

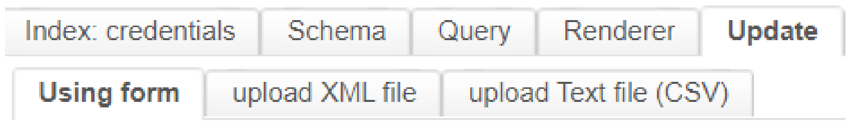
Authentication

Enabling authentication here makes it so that the end user must login to access the search interface, and if access rights are stored for the documents, only the allowed results will be displayed. In this case the authentication type used is the simplest one available: user credentials are stored in an index. The next chapter will cover adding and modifying credentials.

6. Update

For the main index, accessing this tab is not necessary. For the *credentials* index, however, this tab is used to add and modify users. The credentials index contains three fields: *username*, *password*, and *groups*. The *username* field is both the unique and the default field. The password is encrypted using the default analyzer. The user can belong to many groups, so the *groups* field should be a term vector.

Select the *credentials* index from the *Index* tab and then select the *Update* tab again.



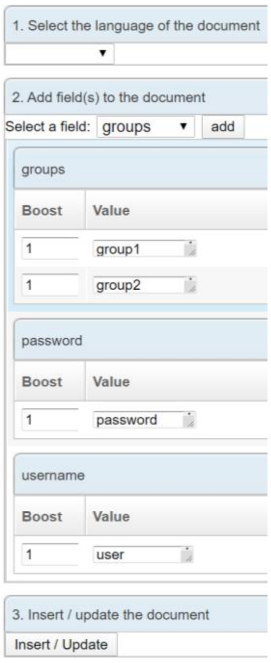
There are three ways new users (or “documents” in general) can be added manually: with a form, an XML file, or a CSV file. For the latter two, ordinary text files can be used but their contents are formatted differently. Any of the methods should work, but XML is preferred.

(continues)

APPENDIX 2. (continued)

Form

Using the form, fields and their values have to added manually. This is not very handy for adding many users who belong to many groups.


1. Select the language of the document

2. Add field(s) to the document

Select a field: groups add

groups

Boost	Value
1	group1
1	group2

password

Boost	Value
1	password

username

Boost	Value
1	user

3. Insert / update the document

Insert / Update

XML

Credentials can also be uploaded in XML format. Following the example available in the XML tab, here is how the file should look:

```
<index>
  <document>
    <field name=”username”>
      <value>user</value>
    </field>
    <field name=”password”>
      <value>password</value>
    </field>
    <field name=”groups”>
      <value>group1</value>
    </field>
    <field name=”groups”>
      <value>group2</value>
    </field>
  </document>
</index>
```

(continues)

Here an “index” contains “documents”, i.e. users, which in turn contain the appropriate fields. Using XML, it is easier to include any number of groups. Generating XML files from Linux users will be covered later.

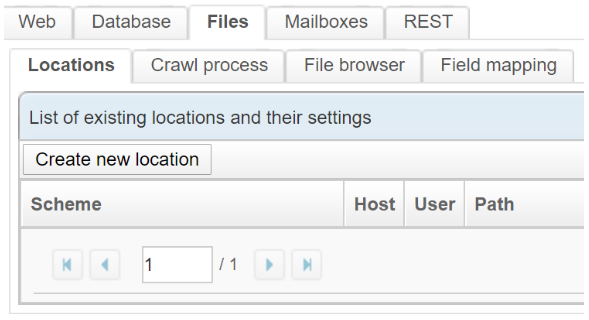
7. Delete

This tab is also not necessary for the main index but can be used to delete user credentials. A query can be executed to check for any matching credentials, which can then be deleted.

For deleting files from the main index, see the next chapter.

8. Crawler

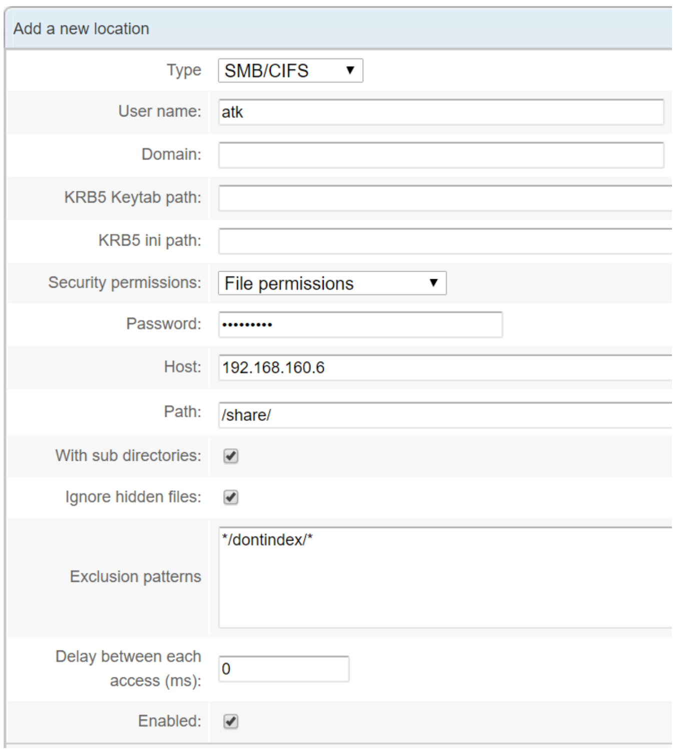
This is one of the most important tabs. Here the crawlers that retrieve files for indexing are defined. Crawlers can be used for a number of use cases, such as web pages, local or remote files, databases, and others. In the case of Samba shares, the focus is on the *Files* tab.



Locations

A *location* can be thought of as a single share that is going to be indexed. These can be defined as follows:

(continues)



Add a new location

Type	SMB/CIFS
User name:	atk
Domain:	
KRB5 Keytab path:	
KRB5 ini path:	
Security permissions:	File permissions
Password:	•••••••••
Host:	192.168.160.6
Path:	/share/
With sub directories:	✓
Ignore hidden files:	✓
Exclusion patterns	*/dontindex/*
Delay between each access (ms):	0
Enabled:	✓

- Type
  - The type of location that is going to be indexed. For Samba shares, use SMB/CIFS.
- Username & Password
  - The crawler needs to use credentials to access the share and its files. A “super user” with access to all of the shares and their files should be used.
- Host
  - This is the address of the machine Samba is running on. It can be *localhost*, 127.0.0.1 or the LAN IP (for example 192.168.160.6).
- Path
  - This is essentially the name of the share. A subfolder could also be specified, of course.
  - Must end in /.
- Exclusion patterns
  - Specific folders or files could be left out of indexing using simple regular expressions.
  - For folders: */dontindex/*
  - For files: */*.txt

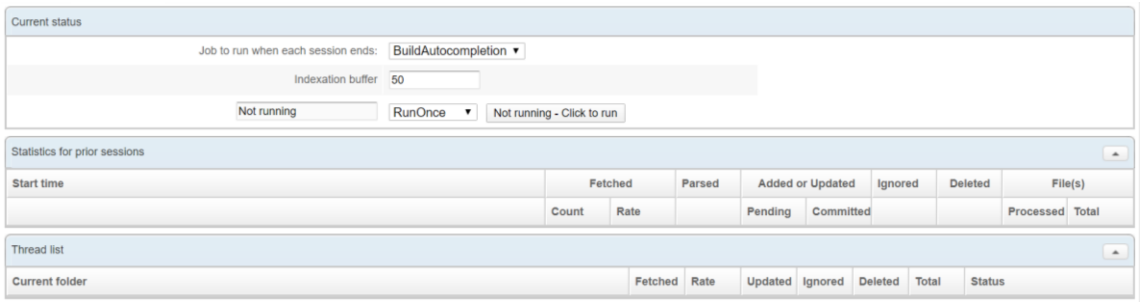
You can check that a location works with the *Check* button.

(continues)

APPENDIX 2. (continued)

Crawl process

After defining one or more locations, the crawl can be started. The crawl can be set to run just once or continuously. By default, a job that builds the autocompletion is ran after each crawl.


Current status

Job to run when each session ends: BuildAutocompletion

Indexation buffer: 50

Not running | RunOnce | Not running - Click to run

Statistics for prior sessions

Start time	Fetched		Parsed	Added or Updated		Ignored	Deleted	File(s)	
	Count	Rate		Pending	Committed			Processed	Total

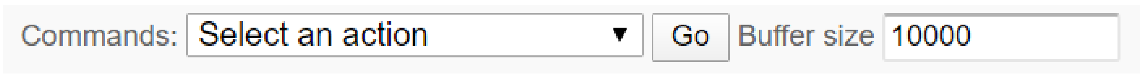
Thread list

Current folder	Fetched	Rate	Updated	Ignored	Deleted	Total	Status

The crawl process can be monitored here. The page refreshes automatically every couple of seconds. It is a good idea to make sure that files are being fetched and committed, though with a large number of files it may be difficult to know whether everything is working correctly. Log files found in the *Runtime* tab may be helpful for this.

File browser

As the files are being processed and indexed, they will be listed here. If no files are listed, try clicking the *Refresh* button in the top right corner of the window. Below the list, a number of commands are available.


Commands: Select an action | Go | Buffer size 10000

- Delete all
  - Delete **all documents in the index**. Also see *Commands* in the *Runtime* tab.
- Delete selection
  - Delete documents currently listed in the file browser.
  - Use filters to list only specific documents.
- Set selection to unfetched
  - Forces the crawler to fetch and update the documents when crawling.

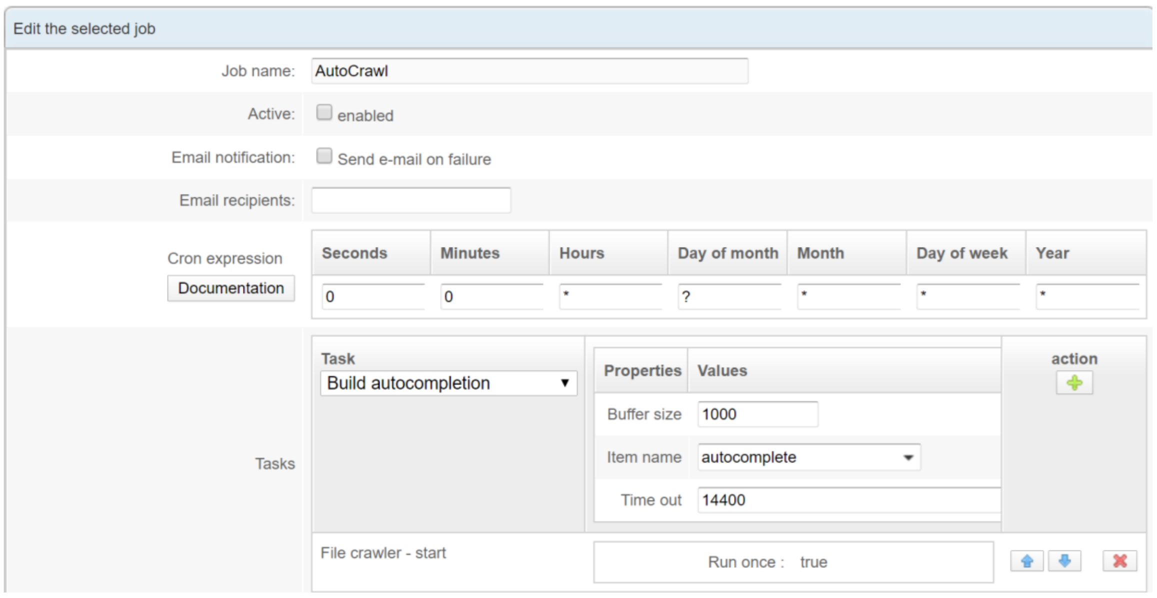
Field mapping

This tab should actually be checked before running any crawls, since it defines which fields of the files are mapped to which fields of the index. The defaults should be just fine, though.

(continues)

9. Scheduler

This tab defines jobs that can be executed at will or at specified dates and times. An arbitrary number of tasks can be added to the job. A job for building autocompletion is defined by default. A job for automatically starting a crawl every hour could be defined like this:



Edit the selected job

Job name:	AutoCrawl
Active:	☐ enabled
Email notification:	☐ Send e-mail on failure
Email recipients:	

Cron expression

Documentation

Seconds	Minutes	Hours	Day of month	Month	Day of week	Year
0	0	*	?	*	*	*

Tasks

Task	Properties	Values	action
Build autocompletion	Buffer size	1000	
	Item name	autocomplete	
	Time out	14400	
File crawler - start	Run once :	true	

The jobs are timed using so-called cron expressions. These expressions can be generated, for example at https://www.freeformatter.com/cron-expression-generator-quartz.html.

Logs are generated for each execution of each job, which can be checked for any errors.

10. Runtime

This tab contains all kinds of information and advanced features.

- Statistics
  - Statistics on various operations executed on the index.
- Commands
  - Commands for reloading, closing, and emptying the index as well as taking it offline and putting it online.
- Cache
  - The cache, if enabled, stores common queries and their results. Can be flushed here.
- Index
  - More statistics on the index.
- Terms
  - Lists of terms indexed for each field. Helpful for determining whether certain documents/folders/fields have been processed and indexed correctly.

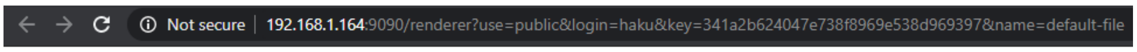
(continues)

- System
  - Information about the system and its resources.
- Logs
  - Past and current log files can be read here.
- Advanced
  - Settings for character recognition, the mail server, the scheduler, and others.
- Threads
  - Information about currently running threads.

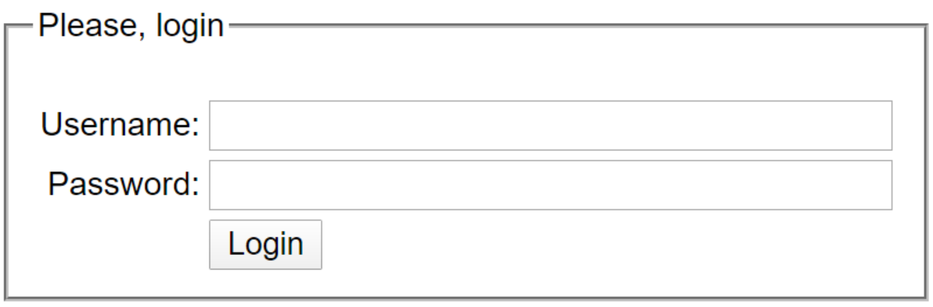
11. Privileges

Here users that can access the admin interface can be created. Having a single admin user and one basic user should suffice. The basic user should only have the right to query the main index.

The end users can access the search interface via the user *haku*, but they also have to log in themselves:



Not secure | 192.168.1.164:9090/renderer?use=public&login=haku&key=341a2b624047e738f8969e538d969397&name=default-file



Please, login

Username:

Password:

Login

Again, note the query strings:

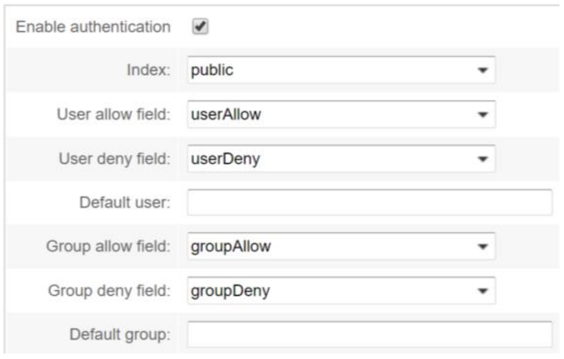
- use=public
  - Name of the index.
- login=haku
  - Name of the user.
- key=[API key]
  - API key of the user.
- name=default-file
  - Name of the renderer.

Now this address, including the query strings, can be shared to the end users in the form of a hyperlink or a shortcut.

(continues)

12. Exporting Linux users

The Samba share crawlers extract file permissions into the fields user/group allow/deny (in reality, the deny field rarely has any values). The permissions are processed (by AccessAnalyzer) to only include the user or group name. OSS checks whether the name or any of the groups of the logged in user match the allowed users or groups and displays the results accordingly.

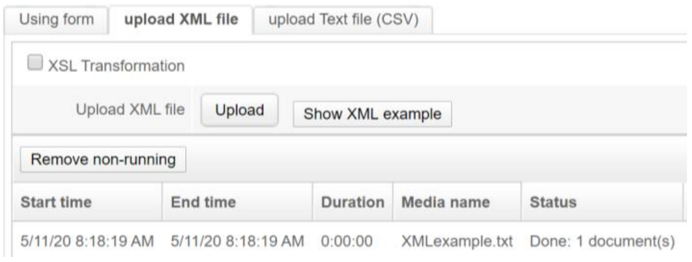


Enable authentication	✓
Index:	public
User allow field:	userAllow
User deny field:	userDeny
Default user:	
Group allow field:	groupAllow
Group deny field:	groupDeny
Default group:	

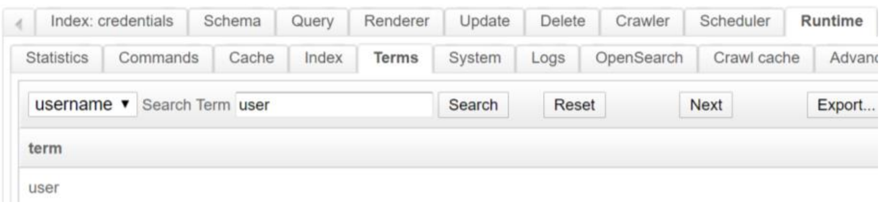
To ensure that each end user has access to the same files as they do in the Samba share, the user’s groups are needed. The script *exportuserxml.sh*, located in the home folder of the admin user, asks for a username and password as inputs, and outputs the user’s credentials in XML format and redirects the output to a text file.

The script does not check whether the user exists or whether the given password is the same as the one set for the user. This is preferable since the password is stored in plaintext in the text file. Ideally, different passwords should be used. After the XML file has been uploaded to OSS, the file **must be deleted immediately**.

The XML file needs to be moved to a location that the browser can access. The file can then be uploaded to OSS via the *Update* tab (into the *credentials* index). OSS should inform that new documents have been added and the user’s credentials should be found in *Terms*, under the *Runtime* tab.



Start time	End time	Duration	Media name	Status
5/11/20 8:18:19 AM	5/11/20 8:18:19 AM	0:00:00	XMLexample.txt	Done: 1 document(s)



term
user