Human resource matching through query augmentation for improving search context

Huy Tran

Tutor, Zlatan Dragisic
Examinator, Patrick Lambrix
Abstract

The objective with the thesis is to research how to match a company’s human resources with job assignments received from clients. A common problem is the difficulty for computers to distinguish what semantic context a word is in. This means that for words with multiple interpretations it is hard to determine which meaning is the correct meaning in a given context.

The proposed solution is to use ontologies to implement a query augmentation that will improve defining the context through users adding suggestions of relevant words. The intuition is that by incrementally adding words, the context narrows, making it easier to search for any consultant matching a specific assignment. The query augmentation will then manifest in a web application created in NodeJS and AngularJS.

The experiments will then measure, based on precision, recall and F-measure, the performance of the query augmentation.

The thesis will also look into how to store document-based résumés, .docx file-format, and properly enable querying over the database of résumés. The Apache based frameworks Solr and Lucene, with its inverted indexing and support for HTTP requests, are used in this thesis to solve this problem.

Looking at the results, the query augmentation was indicated of having somewhat too strict restrictions for which the reason is that it only permits AND conditions. With that said, the query augmentation was able to narrow down the search context.

Future work would include adding additional query conditions and expand the visualization of the query augmentation.
Acknowledgments

I would like to thank company N, where this thesis was conducted, for the opportunity and help that they provided during the course of this thesis.
# Contents

Abstract iii
Acknowledgments iv
Contents v
List of Figures vii
List of Tables viii

1 Introduction 1
   1.1 Motivation .............................................. 2
   1.2 Aim ...................................................... 2
   1.3 Research questions ..................................... 2
   1.4 Delimitations ........................................... 3
   1.5 Method ................................................... 3
   1.6 Thesis structure ....................................... 3

2 Theory 4
   2.1 Semantic web ........................................... 4
   2.2 Query augmentation .................................... 10
   2.3 Data format and hosting ................................. 14
   2.4 Related work ............................................ 17

3 Requirements 19
   3.1 Interviews ............................................... 19
   3.2 Available systems ...................................... 20
   3.3 Intended solution ....................................... 20
   3.4 Requirements ............................................ 21

4 Analysis 22
   4.1 Solr and Lucene ......................................... 22
   4.2 Query augmentation .................................... 22
   4.3 Ontologies ............................................... 23
   4.4 Web application ......................................... 27

5 Design and Implementation 28
   5.1 System requirements ..................................... 28
   5.2 Web application ......................................... 29
   5.3 Solr and Lucene ......................................... 31
   5.4 Ontology ................................................... 31
   5.5 Hosting and SPARQL ..................................... 33
   5.6 Dataflow .................................................... 33
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Precision and Recall</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Precision and Accuracy</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>Human Resource Management Ontology</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>View of HRM ontology in Protégé</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>A graph of the skill ontology by ontograf</td>
<td>25</td>
</tr>
<tr>
<td>4.4</td>
<td>A graph of the skill ontology by OWLviz</td>
<td>26</td>
</tr>
<tr>
<td>5.1</td>
<td>System overview</td>
<td>29</td>
</tr>
<tr>
<td>5.2</td>
<td>A snapshot of the query augmentation</td>
<td>30</td>
</tr>
<tr>
<td>5.3</td>
<td>View of created ontology in Protégé</td>
<td>32</td>
</tr>
<tr>
<td>5.4</td>
<td>A graph of the skill ontology by ontograf</td>
<td>32</td>
</tr>
<tr>
<td>5.5</td>
<td>A graph of the skill ontology by OWLviz</td>
<td>33</td>
</tr>
<tr>
<td>5.6</td>
<td>Dataflow</td>
<td>34</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Results from interviews ................................................. 20
5.1 Domain relations ......................................................... 33
6.1 Results from user tests .................................................. 36
6.2 Results from query evaluation ......................................... 37
1 Introduction

It is currently an age where human resource management is getting increasingly complex where assignments are diverse and consultants’ competencies are filled with a range of different skills, diplomas and experiences. This has resulted in a time consuming task where manually matching an assignment to the best consultant is difficult and in some cases impossible. The reason is not only limited time but also energy which leads to decreased motivation to read through hundreds of résumés and remember them as the sheer amount of information is difficult to process. This process is also error prone as managers may miss information contained in the consultants’ résumés. Furthermore, the information located in résumés is volatile and gets updated regularly with additional competencies creating an unsustainable task for human resource managers. Ability to search, filter and rank the résumés is needed. Additionally, the managers all have their own way of spelling, abbreviations and knowledge about search terms. This is also reflected in the résumés as consultants often have different spellings and definitions of theirs skills. This leads to in-concise queries from human resource managers that, even if it would be correct, would not match a consultant that in reality has skills that would match the assignment.

This problem is recognized by many companies and as of 2015, The Association For Talent Acquisition Solutions reported that there are over 60,000 sites worldwide that offer talent acquisition through matchmaking. To solve this, a couple of solutions has been proposed and researched before such as Human Resource Management Ontologies. Another possible solution is Information Retrieval Systems, Semantic based frameworks where a form of ontology has been used.

This master thesis was carried out at company N that had problems with matching assignments to their employees. N had a résumé bank of over 500 résumés and enough assignments to have an occupancy rate of 98%. N has a search engine today but deem it too primitive and slow to suit their needs. Some employees do not even use the search engine because of mentioned reasons. This thesis will look into another approach where the query is supported and built with the help of an ontology, also called query augmentation, to then be fetched from a database, in this case an inverted index based solution called Lucene.

1http://tatech.org/
1.1 Motivation

N wishes to improve their search engine to better pinpoint a consultant that matches with an assignment. The current process at N enables searching through the résumés to find the correct keyword and is a plausible solution to the problem at hand, but has inherently its own problems that need to be solved. The managers’ search-process and the difficulty to have a computer understand linguistic ambiguity are a few of the mentioned problems. The search term "Java" may refer to the island Java instead of the object-oriented programming language which displays a common mistake that computers do as a result of its binary nature. Managers may believe that they should be able to search with whole sentences while only keyword based search is supported. Wrong conjugation of a word might be used as a search term which might not generate an answer at all compared to using a supported conjugation. One might not be sure what the precise keyword is that will result in someone who fits the assignments’ requirements.

It is also worth noticing that it is rarely the case that the customer precisely knows what kind of consultant that fits the position behind the assignment even if the assignment is detailed. This could lead to assignment requirements omitting words that are crucial to the position or introduce ambiguity. An example of the potential ambiguity is again the term "Java". Even though Java may be interpreted as a programming language, what field the Java consultant needs have experience in is not clear by just the term "Java".

1.2 Aim

To use query augmentation with ontologies to expand the search functionality for guiding the user to include precise keywords. This will in theory narrow down the results to only the ones that match all the keywords that will lead to N spend less time on searching résumés. By focusing on the query building itself separating the database leads to the solution being modular and compatible to any database.

1.3 Research questions

To enable the query augmentation with query building, it is necessary to first solve how to implement the search, ranking and indexing functionality in the résumé data-bank. Without any search functionality it is hard to not only implement but also test the results of the query augmentation. To solve the first problem with the résumé data-bank, the following questions need to be solved.

*How do you extract information for indexation, ranking and searching from the format .docx?*

The résumés are in the format .docx. It is crucial to have document processing to extract the necessary information.

Query augmentation is then feasible to solve when the above question is answered. This implementation needs to take the query construction guidance into consideration for less semantic ambiguity. To solve the query augmentation, the following questions need to be taken into consideration.

*How do we construct the ontology to support querying? Are there reusable ontologies that can be extended?*

A look into how to properly develop an ontology is needed. Suggested terms given by the ontology need to be related to the original search term. It is also necessary to consider querying possibilities when constructing the ontology. There are perhaps already standard-
ized ontologies for skill management.

**How to visualize the query augmentation?**

What approach to visualize the query augmentation should be used? It is important that it is intuitive for the user.

**Considering recall and precision, has the context actually improved?**

It is necessary to assess if the method has improved the context search at all.

### 1.4 Delimitations

A lot of research has gone into creating an optimal ontology where the structure highly supports extensions and automatic generation of ontologies is a hot topic. Although creating an ontology will be a process in this thesis, automatic generation of the ontology is not a part of the project. Additionally, other optimizations such as ontology query optimization will not been taken into consideration as speed is not the main focus in this thesis.

This thesis will additionally to query augmentation also discuss and present *full text search* necessary for querying the résumés. Performance boosts through Field-restrictive search, improved search algorithms and such are important aspects but will not be investigated any further.

The information in the résumés are also strictly confidential. Analyzing the résumés structure and extract and infer relations and information from the analysis would possibly help with creating the ontology. But as it is confidential information, no deeper analysis of the résumés themselves were performed in this thesis.

### 1.5 Method

An agile method is going to be used during this thesis. It will start with a thorough prestudy that helps with defining the requirements. The requirements will then be analyzed and become the basis of the design and implementation of the tool. This will then be reviewed together with a supervisor at N where new requirements will be set and analyzed to then be designed and implemented. When the tool is deemed mature enough, an evaluation based on user and performance testing will be conducted.

### 1.6 Thesis structure

The remaining of the thesis is organized as following. The general motivation and goal of the thesis are presented followed by necessary look into parts such as the semantic web, inverted index, parsing of the file-format .docx, query augmentation and evaluation. This will give an understanding and reasoning behind many of the chosen methods explained in the following chapters. A *requirements* chapter will present a prestudy and system requirements. This will then be analysed in *analysis* followed by a design and implementation in *design and implementation*. The system is then evaluated in *evaluation*. This will all be reviewed in the discussion section that will not only deliberate the results, but also show possible different solutions that could be implemented. This thesis will then finish with a conclusion where a wrap up of the thesis will be presented with examples of possible further work that can be done for improvement. An appendix is attached at the end of the thesis and contains used questionnaires.
Implementing a query augmentation based on an ontology inherently requires coverage of numerous subjects related to the Semantic web. Semantic web tries to redefine the current web and give it more structure, compared to the current web, thus providing a myriad of tools and extensions that need to be investigated. In addition to this, it is also necessary to look into IR, (Information Retrieval). All of this will shape how the construction and maintenance of the ontology will be carried out.

The theory will then continue with the Query augmentation itself. A couple of necessary parts that needs research are the approach context based search, the query language SPARQL and measurements such as Recall, Precision and F-measure. This is to give knowledge and tools to how to implement the Query augmentation. It will also provide methods that enable evaluation of the implementation presenting an indication of the validity of the Query augmentation.

A third important part will be dedicated for research on the formatting and hosting the résumés. This is a necessary step as the résumés contain all the data that can be queried. Without a solid approach for storing that data, the query augmentation will be not only hard but perhaps impossible to implement.

It is important to note that the query augmentation will result in a tool that will aid N when searching for consultants. Given the nature of Semantic web, the tool will be hosted as a web application with a corresponding backend and frontend. Even the ontology and the résumé data will be hosted on web servers. Even though the web-development can arguably be considered as an important part of the tool, the essence of this thesis is still based on the query augmentation and ontology thus making any theory related to HTTP, web servers, proxies etc. that are necessities for the web application, to be omitted.

2.1 Semantic web

The Semantic Web is according to the World Wide Web Consortium (W3C) an extension to the current web and encourages a common cross-platform framework for sharing information. This is mainly done through the Resource Description Framework (RDF) which is designed to present information as a model of meta-data. Other frameworks that are specifically designed

1The main international organization that sets the web standards.
for data that are also used are Web Ontology Language (OWL) and the Extensible Markup Language (XML). This extends the current web of hyperlinked human-readable web pages to making machine friendly thus gaining the analytic powers of computers. The current web is written in Hypertext Markup Language (HTML) and although being a convention widely used on the web, has limitations in its ability to express relations between the pieces of information that it displays. What HTML is good at is placing information on a document but the lack of expressiveness on relations between information makes it hard for machines to derive and come to a conclusion about the information in the document.

The RDF, OWL or XML solve this problem by introducing additional markup within the document that describes the content of the document. This enables managers to add meaning to the content of a document that makes it easier to understand for a human and also machines are able to process the meanings and use human deductive reasons and inference to help with information gathering and search.

A HTML web page without any semantics in it could look like:

```html
<item>Gamla Stan</item>
```

Its semantic counterpart could look like:

```html
<item rdf:about="http://stockholm.se/Gamla-Stan">Gamla Stan</item>
```

2.1.1 Information Retrieval

Information retrieval, abbreviated IR, is a process of obtaining information that are relevant to the requested information. Compared to traditional SQL-queries, that return information that match precisely with the query, IR systems return information that may or may not match the inputted query. Instead it gives a rank on how well different information match the query and is able to sort the information based on relevance [6]. The search can be based on metadata or on full-text indexing as data today may have various forms such as hypertext, semi-structured data and multimedia data like videos and images. This leads to IR being used in search engines where a lot of emphasis is put on systems supporting queries that are loose and not required to have exact precision. [7]

Information retrieval has different strategies for retrieving information. Every strategy needs a different model/structure of data to be effective. Some common and simplistic models are algebraic, probabilistic, feature-based or set-theoretic models.

2.1.2 Description logics

Description logics, abbreviated as DL, are knowledge representation languages that model real world domains to a symbolic model with concepts, roles and individuals [10]. The symbolic model enables computerized systems to understand and analyze the real world that it represents. DL describes relations between many individuals and can arguably be compared with to the object-oriented method of displaying a database’s relations through tables [6].

```html
class FUNCTIONAL_PROGRAMMING is-a PROGRAMMING_METHODOLOGY with languages [0, 10]: LANGUAGES
```

The example above represents that every instance of FUNCTIONAL_PROGRAMMING must be an instance of PROGRAMMING_METHODOLOGY where FUNCTIONAL_PROGRAMMING’s attribute language have between 0 and 10 individuals. Every individual must then be an instance of LANGUAGES. This creates a mathematical basis
that is used by the Semantic web. One of the technologies that uses description logic is for example the Web Ontology Language (OWL) that is a language that is built upon the XML standard for objects RDF.

Various methods exist that could then analyze knowledge bases. Some of them are named in Borgida et al. [9] such as:

- *Reason* - if a relation holds in terms of the attribute and other relation-restrictions that were previously set.
- *Recognize* - if any relation is inferred by the chain of relations and conditions.

### 2.1.3 XML and RDF

Extensible Markup Language, XML, is a markup language that is created for readability for not only humans but also machines [32]. It is widely used as a storage for abstract data that supports any language thanks to its *unicod character* compatibility which is reflected by the amount of document formats that use XML syntax such as Microsoft Office, RSS, and OpenDocument. XML organizes data in a strict hierarchical structure and provides control over the contained information through *tags* and *attributes*.

XML is embraced by the web as a common use for interchanging data over the Internet. W3C describes XML in [32] such as:

"Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere..."

Resource Description Framework, RDF, was first created to store meta-data but has become a standard model for data interchange on the Web [25]. Compared to XML, RDF does not have a strict hierarchical structure but instead provides a loose structure with the so called *triple*. A triple is composed of a subject, predicate and object that combined define relations between terms. This makes RDF able to combine structured and semi-structured data to then be shared between different applications [25]. RDF further simplifies the distribution of RDF data by isolating a triple's definition from other triples contained in other ontologies through *Uniform Resource Identifier*, also known as *URI*. This give RDF modularity which enables re-usage of ontologies even if the terms are syntactically the same [25].

A triple could look like the following:

```
http://myontology#Object_oriented_programming | rdfs:SubclassOf | http://myontology#Programming_language
```

In the above example, it displays that *Object_oriented_programming* is a *SubclassOf* a *Programming_language*. This is not just readable for humans but also is possible for computers to understand. By using the URI of http://myontology, it is possible to use different kinds of predicates instead of rdfs or get information from myontology to aggregate with other data. A demonstration of this:

```
http://myontology#Object_oriented_programming |
http://NOTmyontology#has_language | http://NOTmyontology#Java
```

The above example takes the term *Object_oriented_programming* from myontology and connects the data with NOTmyontology.
2.1. Semantic web

2.1.4 OWL

OWL, Web Ontology Language, is the language that is used for creating ontologies and is an essential part of the Semantic Web. OWL is a based on RDF where the *triple* is the primary way of writing in OWL [11] which gives OWL all the flexibility with the combined structured and semi-structured data collections mentioned in 2.1.3. The W3C even states that the RDF *syntax* is required to be supported by all tools using OWL [11]. It is possible to create ontologies by only using RDF but OWL introduces additional expressiveness compared to the RDF triples. Some additional expressiveness are:

- Sets such as unions and intersections
- Class and property equivalences
- Property restrictions
- Class axioms
- Version information

OWL could be compared to the class hierarchy in object-oriented programming where the subject and object could be considered a representation of nodes while the predicate represents an edge. One difference between object-oriented hierarchy and ontologies written in OWL is the flexibility. Object-oriented class hierarchy is often static and rarely updated as its scope is often limited to relative smaller applications. OWL on the other hand needs to take into consideration the fast moving landscape of Internet which updates its data more frequently than regular applications. The ontology will be updated constantly which OWL needs to take into consideration thus making OWL more flexible when handling frequent updates [19].

2.1.5 Ontology

Ontology, sometimes also referred as a taxonomy, comes from a branch in philosophy known as metaphysics and is a model that represents a domain of knowledge through primitives and is primarily written in OWL [11]. The ontology primitives are typically classes, attributes and relationships, that also includes information about the primitive’s meanings and constraints given by the expressiveness of OWL. The classes classify the terms in the ontology with corresponding attributes, meanings and may also include class axioms and restrictions on what an instance of the class needs to define. The relationships define the relations to other terms in the domain where constraints define the possible usage of the term. This representation collectively describes the whole domain as triples making use of the combined structured and semi-structured data.

A definition of ontology is described in [23, p.1964] and states the following:

" An ontology defines (specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain.

- The specification takes the form of the definitions of representational vocabulary (classes, relations, and so forth), which provide meanings for the vocabulary and formal constraints on its coherent use.

Gruber also discusses the relation between ontologies and taxonomies and what separates them in his published paper in 1993 [17] and states:
"Ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions — that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world. To specify a conceptualization, one needs to state axioms that do constrain the possible interpretations for the defined terms."

Ontologies is one of the standards in W3C where it defines the vocabulary used under the semantic web standard and is one of the building blocks of Semantic Web. Ontology in W3C's Semantic Web standard plays the role as a standard model for data structure that helps data integration where there is ambiguity between data sets. Ontology also tries to standardize and organize large knowledge bases that needs to be shared between multiple sources. Common use cases for ontologies are organizing large semi-structured information such as newspapers.

One of the biggest fields that uses ontologies is the biomedical field with its Gene Ontology GO. GO is a vocabulary that describes the genes and gene attributes in a molecular, biological and cellular level for any organism. As of 2010, it includes 26,514 terms describing a wide variety of different animals.

2.1.5.1 Ontology Engineering and Construction

A critical part in ontologies is the construction of the domain building not only the triples but also the needed constraints on the classes themselves. One of the biggest problems is that the ontologies often get large with the amount of possible relationships between all the individuals. Not only is it necessary to take into consideration which words and information should be held in the ontology, but also what semantic relationships and concepts are needed to fully display critical relations. Another point that needs to be taken into consideration is maintainability as the ontology, being prone for updates, needs to be consistent throughout. Relations, constraints and axioms need to be obeyed or there will be a risk that the contained data may cause conflicts. Additionally, having the Internet filled with customized ontologies might not be optimal either as migrating and merging ontologies contained information might prove to be a difficult task. There is also the aspect of "re-inventing the wheel" when creating an ontology that might have been done before by someone else. Creating ontologies that satisfy everything that has been mentioned can be quite a task which can be observed for the mentioned GO ontology.

The mentioned GO ontology is crafted and maintained through rigorous efforts of manual editing by human curators. Every year, over 400,000 new articles enter the biomedical literature and the organizations rely heavily on the curators to create and update the ontologies as it is crucial to research that relies on the database. This process is not only costly and slow, where it is required to reach a consensus between many experts, but also error prone due to human mistakes which was observed by Gil Alterovitz et al in .

One solution that tries to mitigate the problems surrounding the creation of ontologies, especially targeting the "re-inventing the wheel" and information merging problem, is to create a common standard and methodology for ontologies that tries to accomplish similar tasks. Mike Uschold et al writes about the topic of reusing ontologies in the paper Ontology Reuse and Application. They conclude that reusing an ontology requires considerable effort to understand the ontology with its choices when implemented. This was considered to be quite time-consuming as the ontology implementation might not be trivial. They also mention that the intended reusable ontology might not suit the necessary requirements fully which can give complications further down the road. But even when considering the possible setbacks, Mike Uschold et al concludes that using reusable ontologies is overall cost-effective compared to designing an ontology from scratch.
A solution to the non satisfiable reusable ontology is proposed by Mariano Fernández-López et al in the paper *Overview and analysis of methodologies for building ontologies* [14]. Mariano Fernández-López et al presents a process that re-engineers ontologies containing the steps of *Reverse Engineering*, *Restructuring* and *Forward Engineering* [14]. The *Reverse Engineering* main objective is to put forth a possible conceptual model of the code that the ontology is based on. *Restructuring* tries then to reorganise the model into a new conceptual model that would better suit the new need. *Forward Engineering* then creates a new ontology based on the conceptual model given by the two steps before [14].

Fernández-López also mentions some approaches that are feasible when there is a need to design and engineer an ontology from scratch. This could be that the data is unique enough for it to have no reusable ontologies that matches it’s planned usage. Fernández-López describes *Cyc*, *Methodology of Uschold and King*, *Methodology of Grüninger and Fox*, *The KACTUS approach*, METHONTOLOGY and SENSUS-based methodology [14].

Natalya F. Noy et al in their publication *Ontology Development 101: A Guide to Creating Your First Ontology* writes about how to first design a new ontology from the ground [22]. The publication presents a few steps that should be taken into consideration when developing an ontology.

1. Step 1 - Determine the domain and scope of the ontology
2. Step 2 - Consider reusing existing ontologies
3. Step 3 - Enumerate important terms in the ontology
4. Step 4 - Define the classes and the class hierarchy
5. Step 5 - Define the properties of classes—slots
6. Step 6 - Define the facets of the slots
7. Step 7 - Create instances

### 2.1.6 Protégé

Protégé is a tool developed at Stanford for creating and developing knowledge based solutions. It is open source and has a strong community with a wide extensions and libraries. The following is extracted from Protégés’ own website. [27]

"Protégé’s plug-in architecture can be adapted to build both simple and complex ontology-based applications. Developers can integrate the output of Protégé with rule systems or other problem solvers to construct a wide range of intelligent systems. Most important, the Stanford team and the vast Protégé community are here to help."

Protégé allows building of ontologies without any interaction with the XML and RDF itself which simplifies the creation and maintenance of the ontology which in the end saves time.

The default tools for creation are quite sufficient for building and structuring ontologies. Additional tools that Protégé provides are SPARQL query interface for querying the ontology in a simpler manner. Protégé also includes a DL query interface that uses description logic. Other recommended tools are OWL Viz and OntoGraf for easy visualization of the ontology itself.

Protégé is used to both populate and create the new ontology. The tools in Protégé give a better overview and simplify insertion that would otherwise be written manually to the ontology schema.
2.2 Query augmentation

Query augmentation, also called dynamic querying, is the process of helping and guide users when searching for information where the data set is large resulting in many matches that may not be relevant. Query augmentation involves a visualization of the query construction, where different parameters can be tweaked through intuitive interactions. The interactions can be anything from pressing buttons to have system generated suggestions that help the user to have a better understanding of how the fetch the necessary data while keeping the complex building of the query hidden. This means that a user can manipulate the data presentation without any necessary knowledge of the database or the query language itself. A few examples are the real estate broker system mentioned by Shneiderman \[26\] and SQoogle mentioned by Gruber in \[23\].

According to Shneiderman \[26\], the following principles need to be considered when constructing a query augmentation.

- A visual presentation of the query’s components.
- A visual presentation of the results.
- A rapid, incremental, and reversible control of the query.
- A selection by pointing, not typing.
- An immediate and continuous feedback.

In Shneiderman’s empiric study where 18 psychology undergraduates were observed to compare a query augmentation to natural-language queries, they found that not only did the query augmentation reach the desired answer faster but also gave the users a better experience and more satisfaction. The approach lets users discover the data in a quicker and safer manner.

2.2.1 Context search

A common problem when querying with words is that some have multiple meanings. Without putting a word into a context, it is quite difficult to determine which one of the different meanings the query word should have [2]. This can result in query matches that are non relevant to the real intent of the user. Query augmentation is here a possible solution to the problem. In addition to the user guidance mentioned above, query augmentation may also help with adding the correct context to a query. This gives guidance to the users’ desired results where there are words with semantic ambiguity through a so called contextual search [2]. While context does not have a standard definition [15], according to the dictionary Merriam-Webster Online, there are mainly two definitions of the word context [4]. The first is the parts of a discourse that surround a word or passage and can throw light on its meaning where the second is the interrelated conditions in which something exists or occurs. Both indicate that the context helps with putting words with semantic ambiguity in its supposed meaning in specific situations. There are different approaches to implementing context search [5].

2.2.1.1 Personalisation

Personalisation utilises the history of queries that has been sent to a system to predict future searches. By analysing the query, context is able to be extracted and saved for future use. This has also been seen in recommendation systems where user recommendations are partly based on items that other users, who have voted similar as you on previous items, have bought.

\[2\]http://www.merriam-webster.com/dictionary/context
2.2. Query augmentation

2.2.1.2 Link analysis
Link analysis is an approach that tries to define a context based on information spaces that contain lots of links. An example of this information space is web pages and its different links to other web pages. Through analysis of the links, context may be able to be derived.

2.2.1.3 Language models
Language models are models based on statistical language modelling (SLM). The model is computed through the process of analysing the documents in the data collection. Through the analysis, the model is able to predict what is the most probable word that would follow the already inserted words.

2.2.1.4 Ubiquitous computing
Ubiquitous computing relies on the surroundings to help defining context. One prime example of this is geolocation.

2.2.1.5 User background
User background is an approach that takes into consideration the experience level of the user. New users often need more help with the query than experienced user need.

2.2.1.6 Context of knowledge resources
Context of knowledge resources relies on understanding the information resource itself. Even though knowledge is considered something that is context free, knowing what kind of information resource is used gives hints on the context that should be applied.

2.2.1.7 Setbacks of query augmentation
Even if the advantages are many, there are disadvantages that need to be taken into consideration when implementing a query augmentation. The main road bump is performance. The stress on the system is higher since the data fetching increases. This stress rises even further when many users uses the system at the same time and also if the interactive parts increases [26].

In addition to the performance problems, it is necessary to tailor the interface for query augmentation. The available visualization needs to be not only visually pleasing but also intuitive for the current usage.

2.2.2 SPARQL
SPARQL is an RDF query language that consists of triple patterns to read and write data that follows the RDF specification given by the W3C recommendation. The triple query makes use of the "subject-predicate-object" pattern that combined describes the relations between the data stored in the entire database and enables querying conjunction, disjunction and additional pattern recognition. As SPARQL is a RDF querying language, it is able to use all the possibilities RDF enables such as the Uniform Resource Identifier, URI. This enables querying over multiple RDF data-sets at the same time and aggregate the data to represent a results with higher precision and achieve a more complete view [18]. A simple query fetching the name of everyone who lives in Springfield would look similar to the following:

```
PREFIX data: <http://xmlns.com/data>
SELECT ?name
WHERE {
  ?name data:lives_in data:Springfield
}
```
2.2. Query augmentation

Where the data would look like:

Maggie lives in Springfield
Homer lives in Springfield

2.2.3 Evaluation

After describing both the design elements and querying tools, it is necessary to know relevant measurements on how well the ontology performs.

2.2.3.1 Precision & Recall

Precision and recall are units of measurement that show an indication of how well a system performs when querying for relevant documents [13]. Given a set of documents divided into relevant and non-relevant documents, precision measures how many of the documents retrieved by a search are relevant. Recall on the other hand measures how many of the relevant documents in the whole set of documents get retrieved.

The strict definition of these measurements are:

\[
\text{Precision} = \frac{\text{relevant documents} \cap \text{retrieved documents}}{\text{retrieved documents}}
\]

Recall, sometimes also known as Sensitivity or True Positive Rate (TPR), is similar to precision but looks at how many of the relevant documents are actually retrieved. A high value of recall would indicate that most of the documents in the retrieved set coincide with the relevant set. Again, the recall value by itself could be misleading as a high value can be reached by retrieving all the documents. As recall only checks if the documents from the relevant set are contained in the retrieved set, retrieving the whole set would inherently also retrieve all of the relevant documents resulting in recall getting the value of 1. Recall by itself may give a skewed result [13].
2.2. Query augmentation

A combination of high values of both recall and precision would indicate a good retrieval \cite{13} which is why they both are often grouped and together produce a new measurement called \textit{F-measure}.

Trying to raise precision often lowers recall \cite{13}. F-measure is a combination of both precision and recall, and is a harmonic mean of both the measurements. It tries to give a better measurement of “effectiveness” than recall and precision alone are able to accomplish. Generally, “effectiveness” is defined as

- Precision\footnote{13}. Ability to get results that are closely grouped.

- Accuracy - Ability to get results that are as close as possible to the correct one.

- Trueness - Average of all the results closeness to the target.

An illustration of this can be seen below

\footnote{3}{Precision here is different from the \textit{precision measurement} explained in precision & recall. Any mention of precision from here on will only be of the \textit{precision measure}.}

---

\cite{13} https://upload.wikimedia.org/wikipedia/commons/thumb/2/26/Precisionrecall.svg/640px-Precisionrecall.svg.png?146349199083

---

\cite{24}
2.3. Data format and hosting

2.3.1. File-format

The .docx file-format is one of many document file format that store binary information in a human readable format. Document file formats are basically for storing text based information on computers and there are several standards that have been developed. Some of these standards are PDF, HTML and Microsoft Word (.doc).

Until Microsoft Word 2007, the .doc file-format was used as the standard for documents. With the release of Microsoft Word 2007 a shift into an open standard format took place where the format now is based on XML (Extensible Markup Language) and was renamed .docx where the X comes from XML. This gave developers necessary tools to work with the new format and gave the documents new features that could not been implemented with the old format [16].
2.3.2 Search

Searching small sets of documents is often possible with just a direct scan of the content itself, which is also known as serial scanning. But when the sets of documents get larger, or the number of query terms increases, it proves to be necessary to introduce indexing of the documents that helps with the retrieval speed [21]. The method used when indexing is what is called inverted index that creates an entry for each word found in a document while ignoring, when it comes to searching, often occurring keywords such as “or” and “the”. These non essential keywords are also called stop words. Additionally to removing the stop words, some indexers also use stemming which transforms conjugated words to their basic forms. Both stop words removing and stemming lead to an increase in precision and recall that will be discussed further into this section.

2.3.3 Lucene

Originally written by Doug Cutting in 1999, Lucene is today a free and open-source information retrieval software library. Lucene, written in Java, joined the Apache family in 2001 and has since been ported to other programming languages such as C#, C++, Python, Ruby and PHP. Lucene is commonly used in applications that utilise full text search where Lucene contributes with its full text indexing and searching capabilities [20].

A few of the features that are listed on the Lucene homepage [28] are:

Scalable, High-Performance Indexing
- over 150GB/hour on modern hardware
- small RAM requirements – only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30

Powerful, Accurate and Efficient Search Algorithms
- ranked searching – best results returned first
- many powerful query types: phrase queries, wild card queries, proximity queries, range queries and more
- fielded searching (e.g. title, author, contents)
- sorting by any field
- multiple-index searching with merged results
- allows simultaneous update and searching
- flexible faceting, highlighting, joins and result grouping
- fast, memory-efficient and typo-tolerant suggesters
- pluggable ranking models, including the Vector Space Model and Okapi BM25
- configurable storage engine (codecs)

2.3.3.1 Model data

The Lucene data model consists of documents, fields and indexes. The Lucene document consist of fields where each field has its own name with corresponding unformatted and unstructured text. The index is then a persistent storage where the documents are stored for retrieval. Prior to indexing, a document needs to go through a process that transforms the document into a Lucene document. The process consists of three steps:
2.3. Data format and hosting

1. Parse the field content into a stream.
2. Tokenize the stream.
3. Apply operations on the tokens.

A "token" in this context is the definition of a single word or atomic parsed element. An "operation" usually consists of lowercase transformation, removal of stop words and stemming \[20\].

2.3.3.2 Querying

When querying the data storage, Lucene’s own query language can be used. It supports:

- **TermQuery** - Consists of one term composed by two elements, the to be searched field and the text expression being searched for.
- **RangeQuery** - Querying in a lexicographically order with a range of terms.
- **PhraseQuery** - Uses the positional information on the index to locate terms that are within a certain distance.
- **WildcardQuery** - Searches for contents with expressions containing wild-cards such as (*).
- **FuzzyQuery** - Tries to match terms’ similarity based on the Levenshtein distance algorithm.
- **BooleanQuery** - Uses boolean operators such as AND, OR and NOT.

20

2.3.3.3 Scoring

Lucene scores the document based on a variant of the \textit{Tfidf} scoring model \[5\]. The factors that need to be taken into consideration are the following:

1. tf - term frequency in document = how often the word appears in the document.
2. idf - inverse document frequency = how rare the word is.
3. coord - number of words in the query that were found in the document.
4. lengthNorm - measure of the importance of a term according to the total number of terms in the field.
5. queryNorm - normalization factor so that the queries can be compared.
6. boost (index) - any boost of the field at index-time.
7. boost (query) - any boost of the field at query-time.
2.3.4 Solr

Solr, also called Apache Solr, is written in Java and built on Apache Lucene. Solr provides a full text search server where it uses Lucene for indexing and search. Yonik Seeley created Solr in 2004 and merged with the Apache Lucene project in 2010.

Not only is it possible to search but also gain additional information such as hit highlighting, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features and document (such as .docx) handling. The Solr server provides a REST-like API that simplifies HTTP-based queries. The results can then be received as JSON, XML CSV or binary results [4].

Features listed on Solr’s own homepage [4] are the following

- Advanced Full-Text Search Capabilities
- Optimized for High Volume Traffic
- Standards Based Open Interfaces - XML, JSON and HTTP
- Comprehensive Administration Interfaces
- Easy Monitoring
- Highly Scalable and Fault Tolerant
- Flexible and Adaptable with easy configuration
- Near Real-Time Indexing
- Extensible Plug-in Architecture

Using the indexing and searching capabilities that Lucene provides, Solr also aids with the result renderer which orders the result items before sending them to the user [1].

2.4 Related work

Already back in 1975, Moshe M. Zloof at IBM saw a trend where query languages were created to appeal to non-professional users. The ”Query by example” Language is presented through illustrations and examples [33]. The query is built through examples where the user fills in appropriate table rows with examples of answers that the user wish to get returned. By using simple syntax such as words without underlining are constants and underlined words are variables. Other supported features are groupings and customizing the output for proper parsing. This enabled the language to handle rather complex queries without demanding the users being professionals.

In Semantic matchmaking for job recruitment: An ontology-based hybrid approach [13], Maryam Fazel-Zarandi et al recognizes a trend where companies, especially consulting firms and software development companies, need to understand and have a sense of the human resource competency. A description logic is used for reasoning about the job postings and applications aiming to break down the skills and then populate the ontology. The description logic also tries to infer information that takes into consideration other aspects such as competency, degree and work experience. When matching an applicant to a job posting, the paper defines different levels of requirements to help with ranking in case of multiple matching applicants. The paper resulted in an approach that has both high precision and high recall while taking into account the degree of similarity in a match.

Aljadda et al [2] proposes a different solution to the implementation of query augmentation. The paper discusses the usage of both inverted index and ontologies, and pushes the notion of using crowd sourcing as a base for query augmentation. Through the usage of query logs of over a billion user searches, there is a higher chance to find relations between two different words. The higher the number of users that search for a set of words,
the higher the chance that there is a relation between those words. By reviewing users validity, it is possible to minimize noise improving the accuracy of the inferred relation. Aljadda et al discusses the utilization of lexical database and domain-specific taxonomies. A human expert will be in control and specifies a list of relations between words. The thesis states that the needed manual work for the lexical database is quite taxing and needs to be repeated on a language-by-language basis. Another discussed option is Latent Semantic Indexing (LSI) that attempts to automatize the finding of the semantic relationships. This is a possible option but tends to give an inaccurate result leaving users receiving unexpected results. A third option is discussed where Collaborative Filtering (CF) is used. Collaborative Filtering uses links that are formed when a user interacts with two documents after a query. The notion of a user selecting the documents during the same query is an indication of the relevant relation between them. The content itself is not used for deriving relations.

Solr and Lucene are both well documented and used tools for storing and querying documents in the .docx fileformat. Having that said, Solr is not the biggest or most used tool out there. The most used one is Elastic search which is an alternative to the set up proposed in this thesis [12].
3 Requirements

Before beginning creating the new system, interviews with the managers were conducted to understand the work process and extract any necessary requirements from the managers themselves. The interviews will also give a hint on what the current work process lacks in terms of functionality which also helps with setting the requirements for the new system.

Even though some understanding of the system will be obtained from the interviews, it is necessary to make a thorough research on what systems actually exist so that all the parts are known. Knowing all the already implemented systems helps with understanding the current functionality but also may save time as some of the systems may be reusable.

3.1 Interviews

The interviews were conducted at N with the managers before developing the new tool where seven of the employees were questioned about the current matching process. They took a small test where they had to search for candidates who would match a given description of an assignment. The goal was to understand what is currently used at N and also try to extract keywords from the test that would help with constructing the ontology. The results will showcase how the work process looks like as of today.

The interviews consisted of a questionnaire with questions related to the current workflow and a real situation scenario where the user is presented with an assignment to match with a real consultant. The questionnaire also included the users opinion on the current systems and what the would like to improve.

Table 3.1 shows the results from the questionnaire.
3.2. Available systems

The implemented systems were the Apache Solr and Lucene systems that already had indexed the résumés. This system was in an infrastructure where servers hosted the tools making it quite easy to communicate through HTTP between the systems.

The internal systems are all also web based to make it easier for employees to use the systems without installing a dedicated application on their computers.

3.3. Intended solution

This thesis intends to implement a solution where the ambiguity in context is solved by using query augmentation with ontologies. The ontology will keep track of all the relations between the words and supply them to the query augmentation.

3.3.1 Base for query augmentation

The query will be stemmed from a typical assignment that N receives and the query augmentation. The assignment looks like following:

"Company A looks for a consultant who has deep knowledge in Python and has experience with working with the methodology kanban. The consultant will work"
with the current backend-system in django and will be a part of a team that maintains and upgrades the system to the new needs. The focus for the consultant will be the backend system but the consultant also needs to have worked with new technologies such as reactjs and be able to quickly learn new systems as the members on the backend and frontend teams can be reallocated."

A quick look at the assignment above gives an indication that it is important that the consultant has knowledge in the programming language python with its framework django and is able to work with kanban as both the backend and frontend teams are interchangeable. Some mentions about being able to work with new technologies such as reactjs. This results in a consultant that is a web developer who knows python and django, worked with kanban and is able to work with technologies such as reactjs.

### 3.4 Requirements

Given the available systems, the intended solution and the answers from the interviews, the requirements on the new tool are the following:

- The tool needs to be a web application with a corresponding frontend and backend.
- The tool needs to communicate with already established internal systems.
- Search will be complemented with query augmentation.
- Query augmentation will be based on ontologies.
- Query augmentation will give suggestions on related words.
The main problem that needs to be solved is search through the résumés. Considering the problem, a couple of parts are necessary. First, the managers need to have an interface that they can interact with. To mitigate the problem with ambiguous context when searching, mentioned in 2.2.1, the search will be accompanied with query augmentation. The query augmentation needs to have a simple interface where the words are suggested in an easy and viewable manner. Putting everything together gives a web application that in turn needs to be hosted on a server. The web application server then needs to communicate with the Solr server to get access to all the search capabilities that Lucene provides. In addition, the query augmentation will be implemented with an ontology as it’s base. This ontology needs to be created, maintained and hosted on a server for the web application server to be able to communicate with it.

4.1 Solr and Lucene

Apache Solr and Lucene as mentioned were already implemented before this thesis was carried out. Other options were available, such as the elastic search mentioned in Related work 2.3, but as the focus in this thesis is the query augmentation itself with ontologies, using the already implemented Solr and Lucene saves time and effort. That Solr also supports query through HTTP requests makes it easier to communicate with the system.

4.2 Query augmentation

The query augmentation needs to be constructed with the context search in mind. The interface on the frontend needs to take into consideration the adding of terms to the query and making it seamless for the user to use the system. A method of doing this is, is by looking at one of the most used web-tools today Google. Google is one of the most used search engines today and gives a hint on a good approach to implementing the query augmentation. Google helps users by automatically suggesting words and sentences based on the user inputted query. The Google engine basically tries to guess what the user is looking for by the words the user has typed. Naturally, Google bases the auto-completion on other factors and algorithms than ontologies, but a similar approach should be viable for our web application.
Another argument why the Google approach is a good idea is that based on Ben Shneiderman's principals mentioned in the theory in section query-augmentation [2.2] all points can be checked. A visual presentation of the query's components is satisfied by visualizing the suggestions given by the ontology. The results from the Solr query gives a visual presentation of the results where it can swiftly be adjusted through removing or adding terms to the query either by the users own accord or by using the ontology suggestions. The ontology suggestions can be added with minimal effort by pointing, not typing. In other words, a rapid, incremental, and reversible control of the query where the user receives an immediate and continuous feedback.

SQoogle [23, p. 1975, Figure 1] is a SEWASIE, Semantic Webs and AgentS in Integrated Economies, project and has a similar approach to Google. Here the visualization is the structure of the ontology itself where a starting point sets where in the ontology the visualization will begin at. The user is then able to navigate through the ontology and set restrictions to get the data needed. Although not as complex, the query augmentation in our web application will display a similar visualization but without the extensive capabilities of navigating through the ontology.

4.3 Ontologies

As discussed in the theory part [2.1.5.1] a lot of resources need to be allocated to the creation of an ontology. Not only does it need to be created with maintainability in mind, but it needs to do so with semi-structured data. Using reusable or existing ontologies would be a good solution as the complexity of designing the structure and relations are already made. Not only may it save time but may also make it more portable if the reusable ontology is a standard format. With that said, as mentioned in [2.1.5.1] using reusable ontologies are not without any caveats and may possibly slow down the work if it badly fits the task at hand.

4.3.1 Reusable/Existing ontologies

Considering the limited time, an initial search to look for reusable ontologies that would fit with the skill oriented data that would populate it was done. One created by Boris Villazón-Terrazas et al [30] was found and showed a lot of promise as it was engineered to act as a common language for job posting and résumé collection. The ontology created by Villazón-Terrazas et al is a Human Resource Management ontology composed of thirteen different modular ontologies meaning that it also stored data on other aspects besides the skills such as education. It has a larger usecase where the ontology could be used to process a whole “search and match” between employees and jobs. All of this was not necessary and only the skill-based ontology was extracted and used. Figure 4.3.1 displays the whole Human Resource Management ontology:
The three following figures, 4.2, 4.3, 4.4, show graphs created from Protégé 2.1.6 and display the skill-based ontology extracted from the Human Resource Management Ontology above. They all represent a visualization of the ontology structure with relations and properties.
4.3. Ontologies

Figure 4.2: View of HRM ontology in Protégé

Figure 4.3: A graph of the skill ontology by ontograf
4.3. Ontologies

4.3.2 Change ontology implementation

While populating the ontology, it was quickly realized that the ontology was not well suited for adding relations. The ontology itself is better suited for storing the skills that are related to a person with its hierarchical class structure. Compared to this, what is needed is to add relations between the skills themselves making Villazón-Terrazas et al’s ontology unsuitable for the task at hand. Storage is of course one important aspect but being able to set relations enables simplified SPARQL queries as it only needs to focus on relations in comparison to class hierarchical structure where the SPARQL queries needs to take traverse the hierarchical tree. There is the possibility to re-engineer Villazón-Terrazas et al’s ontology but was deemed as the incorrect way to go as it is clearly was not intended to be used in such manner. Consid-
er the arguments above, it was decided to create a new ontology that was better suited for inserting words in a less hierarchical class structure and apply relations between the words. One of the most prominent standards to create an ontology is described in *Ontology Development 101: A Guide to Creating Your First Ontology* written by Natalya F. Noy et al [22]. Creating your own ontology gives more flexibility as mentioned when setting up relations. This gives an ontology better tailoring to the data-set it should reflect.

The following steps are mentioned in the ontology guide [22]:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy.
5. Define the properties of classes-slots.
6. Define the facets of the slots.
7. Create instances.

### 4.3.3 Ontology hosting and SPARQL

When the ontology is created, it is necessary to host it on a server for querying. The server needs to support HTTP requests and SPARQL which lead to the usage of Apache Jena Fuseki. Apache Jena is an open-source java-based framework for building semantic web applications. It provides helping libraries for handling RDF, OWL and SPARQL. Apache Jena enables storing ontologies on servers where querying with SPARQL triples over HTTP. Apache Jena Fuseki is the SPARQL part of Apache Jena and is a server that provides as mentioned a RESTful api that enables communication by SPARQL queries through HTTP.

### 4.4 Web application

The web application needs to visualize the query augmentation and search capabilities that the application will provide. The application will be implemented as a single-page application which needs to be taken into consideration when choosing the web framework for the front end. Several bigger frameworks exists that are maintained by large companies such as Google and Facebook. Choosing an already established framework saves time as writing the application in pure javascript would be quite time consuming.

The backend also needs to bridge the communication between all the systems such as the Solr server and Apache Jena Fuseki server. It will also connect the application to other necessary parts of the internal systems such as the login. For this, there are several server side programming languages and frameworks where some may be arguably perform better at the task at hand. But as the current usage is not in a scale big enough for it to make a difference, any programming language can be chosen.
5 Design and Implementation

5.1 System requirements

There are a couple of requirements to set up the system. There are two databases that the system needs to communicate with where one of these is the résumé database that contains the inverted Lucene index for all the résumés. The other database is for storage and querying of the ontology. The three necessary servers that connect the application together are the Apache Jena Fuseki server, Apache Solr server and application server. The application server serves the user interface and takes care of parsing the queries and redirect them, through a proxy, directly to the other servers. The Apache Jena Fuseki server has the responsibility of all the ontology related requests while the Apache Solr server controls all the requests to the résumés.

Figure 5.1 shows a visualisation of the system.
5.2 Web application

The following frameworks and tools will be used in the web application.

5.2.1 Frontend

The frontend will be written with AngularJS. AngularJS is an open-source javascript based framework created and maintained by Google. It gives an approach to develop Single-page applications where it provides solutions for model-view-controller (MVC) and model-view-viewmodel (MVVM) architectures. The AngularJS approach has been, as of the time this thesis was written, widely adopted by companies such as Paypal and Netflix. A single-page application facilitates the query augmentation as the requirements are simply a input-field and a list of results, thus making AngularJS a good choice as a frontend framework. Having it maintained by a company such as Google further secures the stability and robustness of the framework.

5.2.2 Backend

The backend will be written with NodeJS and Express. NodeJS is an open-source run-time environment for building large and scalable web servers. It ships with the package manager NPM (Node Package Manager) which provides a plethora of packages and libraries that
helps the building of web servers. Based on Chrome’s V8 javascript engine, the server can be 
written entirely in javascript which gives the backend and frontend a common programming 
language.

Express is one of the frameworks that adds functionality to NodeJS. It mainly helps with 
setting up the server as a RESTful-server where it simplifies the HTTP-routing, creation of 
middle-wares such as JSON parsing.

The NodeJS+Express backend is used for connecting the frontend application with the 
Apache Jena Fuseki server that serves the ontology. It will also connect the web application 
to the Solr server for querying of the résumés.

5.2.3 Query augmentation

The query augmentation itself will be visualized in a similar way as SQoogle and Google. A 
snapshot of this is shown in figure 5.2.

Figure 5.2: A snapshot of the query augmentation

5.2.3.1 Word incrementation

Given the assignment presented in Requirements 3.3.1, an example of how the query aug-
mentation can be presented.

"Company A looks for a consultant who has deep knowledge in Python and has 
experience with working with the methodology kanban. The consultant will work 
with the current backend-system in django and will be a part of a team that main-
tains and upgrades the system to the new needs. The focus for the consultant 
will be the backend system but the consultant also needs to have worked with 
new technologies such as reactjs and be able to quickly learn new systems as the 
members on the backend and frontend teams can be reallocated."

Going with the words appearing in chronological order, the query could look like:

python - kanban - django - reactjs

The query augmentation will start with the word "python" and then incrementally add 
words from there until a similar query emerges. The biggest reason here is to start with a 
base set of résumés as the résumés need to be checked manually for relevancy. Relevant 
résumés are then based on whether the consultant is a web developer or not. At the same
time as limiting the base set, the picked query should still not restrict the set too much as later added words might not impact the results in a significant manner. The relevant résumés will be decided as the ones who are web developers leaving out résumés who have project management, mobile development, IT-security etc.

After each incrementally added relevant word, the list of new keywords will be sent to the Solr server which will query for relevant résumés. The amount of returned résumés is logged and measurements such as Lucenes ranking value will be observed. To determine if the résumés are relevant or not, as mentioned, Relative Precision, Relative Recall and Relative F-measure will be measured.

5.3 Solr and Lucene

The design of Solr and Lucene will be kept intact to the one already implemented at N. Occa-
sional re-indexing is performed but no actual change of the system will be done.

5.4 Ontology

Unfortunately, the reusable ontology created by Boris Villazón-Terrazas et al [30] was not fit for the task at hand. A new ontology will be created from scratch based on the guide given by Natalya F. Noy et al [22] where the following was stated:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy.
5. Define the properties of classes-slots.
6. Define the facets of the slots.
7. Create instances.

The first point, the domain in this thesis is the skills and experiences contained in the consultants’ résumés. Even though only a subset of the skills and experiences are inserted into the ontology, it still is developed with the domain in mind. Reusing an ontology was mentioned in analysis but arguments were given why it is not going to be used. Continuing with point three, iterating over all the résumés was considered a daunting and time consuming task that may not necessarily result in a better outcome. For that reason, a simplification is made where only a subset of all the words are inserted. The tests that will evaluate the query augmentation will be based on an assignment around web development. The ontology will as of that be populated with skills and experiences related to web development. The class hierarchy is created through a bottom-up strategy. This strategy felt natural as the words themselves are given. Identifying the classes and relations from there is a natural task. The properties are then used to represent the relations between the individual words in the ontology where the facets do not have any importance. The instances are then created.

The figures, 5.3, 5.4, 5.5 and table 5.1 show graphs that are created by the same tool as the figures in the skill-based ontologies.
5.4. Ontology

Figure 5.3: View of created ontology in Protégé

Figure 5.4: A graph of the skill ontology by ontograf
5.5 Hosting and SPARQL

The ontology will then be populated with words and skills of consultants in their résumés, from multiple users. The word-set will mostly be based on web-related words as this needs to be done manually where each and every word and relation needs to be set up. Even by using Protégé, this takes a substantial amount of time.

5.5 Hosting and SPARQL

The ontology then is served with Apache Jena Fuseki which provides the RESTful api that can be reached through HTTP. The SPARQL is then used to query the ontology. Through the Apache Jena Fuseki, it is possible to query the ontology with HTTP requests containing a SPARQL query. The query used in the tool is based on AND conditions. The AND conditions only take into consideration words that are related to the current words in the query. This results in the retrieved words from the ontology that follows the same context as the query.

5.6 Dataflow

A use-case of the query augmentation would be that the user inserts a word for searching the databank with all the résumés. This word is then highlighted as a keyword and a request containing a SPARQL query, based on the keyword, is sent to the NodeJS server that proxies it to the Apache Jena Fuseki. The Fuseki server executes the SPARQL query and returns the
result containing all of the related words to the user. The user can then choose to include desired related words, input a new keyword or send the current keywords to the Apache Lucene Solr server. The Solr server then searches through the database after matching documents based on Lucene query process and returns the results to the user.

The dataflow in a simpler and more constructed manner could be described as following:

1. User inserts a keyword (Java).
2. A SPARQL query based on the keyword is sent to the NodeJS server (SPARQL(Java)).
3. NodeJS server has a proxy that sends the SPARQL query to the Fuseki server (SPARQL(Java)).
4. Fuseki server looks up all the relevant words that the keyword is connected to.
5. Fuseki server sends back the relevant words back the the frontend where AngularJS inserts them to the query field (RelevantWords(Java)).
6. User can choose to insert additional words to the query by selecting one of the related words or insert a new keyword leading back to [2].
7. User sends the keywords to the Solr server.
8. Solr server queries the data-bank for résumés matching the keywords.
9. Solr returns the résumé results to the user.

Figure 5.6 displays the data-flow in a graphical way:

Figure 5.6: Dataflow
6 Evaluation

The evaluation consists of interviews and testing conducted on the users at N that mainly are in need of querying the database of résumés. The users were then able to test a prototype of the tool and had to fill in a survey containing questions to collect data.

Another test is held where the terms in a query will be incremented word by word by the query augmentation. Recall, precision and F-measure will measure how well the query augmentation fares.

6.1 User evaluation

The users test the new tool and compare it to the old tool. They will have to give feedback on how good the new tool is compared to the old through how relevant resulting résumés are. The evaluation was conducted through a survey that described the web application. The users had to then try to find adequate consultants to the assignment written on the survey. The resulting data from the user defined query was then documented.

6.2 Query evaluation

The query augmentation is evaluated to determine if it improves the search by fetching résumés as relevant to the assignment as possible. To do this, the query will stem from a keyword and will be increasingly longer by incrementally adding relevant words based of the ontology. The purpose of this is to see how much the added words contribute to the context and possibly lead to better results. Each individual added step will be observed and measured by Relative Precision, Relative Recall and Relative F-measure.

6.3 Measurements

The following measurements are used to determine how well the new tool performs.

6.3.1 Relative Precision

Relative precision will describe how many of the résumés returned are actually relevant. This will mostly be very high as Lucene will only return résumés that contain the keywords. Al-
though the presumably high value, it is still important to check if there are any irregularities in the résumés returned. Compared to the regular precision, relative precision will be measured based on the initial set returned by the first query.

6.3.2 Relative recall

Relative recall is used instead of regular Recall as the amount of relevant documents are unknown making it hard to determine the exact value of Recall. Instead, Relevant recall will be measured based on the initial set, the set returned by the keyword "Python", and be normalized from there. By adding words that are synonyms of each other should give interesting results.

6.3.3 Relative F-measure

Relative F-measure is based on regular F-measure but will use the Relative recall instead of regular recall. This will give better measurement of the true "effectiveness" of the query augmentation as it measures the harmonic mean giving a value more suitable for measurement and comparison.

6.4 Results

The following results were collected during the tests.

6.4.1 User evaluation

The prototype of the tool showcased the functionality that the end application would have but had an incomplete user interface. A survey accompanied the evaluation where the users had to try to again match an assignment with a consultant through the use of the new tool. The assignment given during the user evaluation is the following:


<table>
<thead>
<tr>
<th>User</th>
<th>Query</th>
<th>Results</th>
<th>Impression</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>python, django, continuous integration, mongodb, react</td>
<td>1</td>
<td>Positive</td>
</tr>
<tr>
<td>B</td>
<td>python, django, reactjs</td>
<td>14</td>
<td>Positive</td>
</tr>
<tr>
<td>C</td>
<td>undefined</td>
<td>58</td>
<td>Positive</td>
</tr>
<tr>
<td>D</td>
<td>php react</td>
<td>40</td>
<td>Positive</td>
</tr>
<tr>
<td>E</td>
<td>undefined</td>
<td>undefined</td>
<td>Positive</td>
</tr>
<tr>
<td>F</td>
<td>python, django, continuous integration</td>
<td>17</td>
<td>Positive</td>
</tr>
<tr>
<td>G</td>
<td>django, react</td>
<td>7</td>
<td>Positive</td>
</tr>
</tbody>
</table>

Table 6.1: Results from user tests

1During this evaluation, written in swedish
Most of the managers determined that "django" and "react/js" were the most important keywords in the assignment. Only one user inserted above three words in the query whereas two of the managers did not document what their queries were. All managers had a very positive impression on the tool.

6.4.2 Query evaluation

The query evaluation is, compared to the previous test, not focused on how the tool can be used, but focuses more on how well the tool performs. The query is built from the query augmentation. The chosen measurements are relative precision, relative recall and relative F-measure. Combining it all, it will give an indication on how well the tool performs. The measurements are taken during every step of the word incrementation.

<table>
<thead>
<tr>
<th>Query step</th>
<th>Results</th>
<th>Relevant</th>
<th>RP</th>
<th>RR</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>284/284</td>
<td>153/284</td>
<td>0.54</td>
<td>1.00</td>
<td>0.701</td>
</tr>
<tr>
<td>python, django</td>
<td>63/284</td>
<td>41/63</td>
<td>0.65</td>
<td>0.268</td>
<td>0.380</td>
</tr>
<tr>
<td>python, django, webbutveckling</td>
<td>52/284</td>
<td>39/52</td>
<td>0.75</td>
<td>0.255</td>
<td>0.381</td>
</tr>
<tr>
<td>python, django, webbutveckling, reactjs</td>
<td>3/284</td>
<td>3/3</td>
<td>1.00</td>
<td>0.020</td>
<td>0.040</td>
</tr>
<tr>
<td>python, django, webbutveckling, reactjs, kanban</td>
<td>1/284</td>
<td>1/1</td>
<td>1.00</td>
<td>0.007</td>
<td>0.014</td>
</tr>
</tbody>
</table>

Table 6.2: Results from query evaluation where RP = Relative Precision, RR = Relative Recall, RF = Relative F-measure

The first step with the query "python" sets the basic set with 284 résumés. These were then manually checked and 153 were deemed relevant based on the restriction that the consultant needs to have worked with any kind of web development. Almost half of the résumés were not relevant to the assignment. The next incrementation through the query augmentation adds the framework "django". This cut the returned amount of résumés to 63 where 41 of them were relevant. Continuing using the query augmentation, the next word "webbutveckling" was added to the query. This in turn returned 52 résumés where 39 of these were relevant. The word "reactjs" is added next through the query augmentation. This greatly reduced the amount of returned résumés to only three where all three of them were relevant. At last, the word "kanban" was added manually and not by the query augmentation. This adds a different context to the ones before it where the focus is more on management and work methodology. It cut down the amount of returned résumés to only one which was also deemed relevant. The last résumé contained all the words contained in the query.
The discussion will present thoughts about the results given from the tests done in evaluation 6. The design and implementation will also have its own section where the method and design itself is discussed with possible alternatives that might give an improvement. This will then end with a smaller discussion about the ethical aspects that was taken into consideration during this thesis.

7.1 Results

This section presents the discussion related to the results received during chapter evaluation 6 and also include the interviews presented in chapter 3 requirements.

7.1.1 Interviews

The search engine that $N$ has today is not used at all by the managers. A reason for this is that the current search-capabilities are not entirely known by the managers. Most of the managers do not even know that it is possible to search for consultants’ résumés through a search engine. Many of the users use instead Windows search where they search through the folder where all the résumés are located.

Having the majority of the managers matching assignments to consultants through either looking at the over 500 résumés, ask colleagues or by memory alone may seem like an implausible task. This is only possible as $N$ has a list of the available consultants. Combining the list with $N$'s high employment rate, the amount of available résumés are easily identified and are able to be thoroughly checked. This is of course not a sustainable approach to the problem at hand as the employment rate could change at any time. It also introduces a problem where newer or less notable consultants might not get as much attention, as they might not leave a lasting memory to remember, even though they might be better suited for an assignment.

Having that said, to base one’s decision entirely on what the new search-tool’s results are may also not be the best option. Not only do résumés not always reflect all of the skills and experiences that a consultant have, as some may be left out or the résumé is just not updated, but also important aspects such as personality and interest are hard to represent in text. It could arguably be more important to have a consultant who is interested in taking an assignment rather than having someone who has experience in the subject. This also applies
to new consultants. New consultants are often not as experienced and do not have as many skills written on their résumés. Going solely on the search results would neglect those who have less skills or experience thus making it harder for new recruits to develop and improve. The search results should thus only be considered a guideline on good candidates for an assignment and not as an absolute rule of decision.

7.1.2 User evaluation

Regarding the results from the user evaluation, it is plausible to say that the set is quite small with only seven distinct sources. In addition to that, it does not help that two of the managers did not report what their queries were when using the tool. The test was primarily to understand if the tool was something that was usable. Given that the impressions from all of the managers were positive, it is determined that the tool itself is on the right path. The data on the length of the queries themselves may indicate that most managers might only insert up to three words when searching for consultants. But as stated before, the set of managers is too small to come to such a conclusion.

7.1.3 Query evaluation

The query testing gave results where the measurements Relative Precision, Relative Recall and Relative F-measure are calculated from a base set of résumés given from the query “python”. The relative precision shows a constant increase while relative recall and f-measure are both declining. A possible conclusion to this is that the query augmentation helps with narrowing the search context. This increases the precision while it may lower recall as of the inverse relation mentioned in 2.2.3.1. But considering that the f-measure also lowers, it might indicate that the restriction is too strict. The AND-only condition, mentioned in 5.5, might be the culprit to this. A possible remedy is to add NOT and especially OR conditions as this would lessen the restriction.

7.2 Implementation and design

During implementation and design, a few limitations had to be laid because of time and existing systems’ restrictions.

7.2.1 Ontology

The engineering and development of the ontology itself may be considered simplistic. The hierarchy is not very deep and focuses more on relations and attributes that each individual could have. This enables an easier SPARQL query as the query only needs to take into consideration the relations instead of traversing the tree hierarchy. One disadvantage of this is that adding additional words to the ontology might be quite complex as almost all the relations need to be set up. One could argue that the ontology could be more refined, but in this thesis this would not change the results.

The extent that the ontology is populated is also limited. The amount of skills that are populating the ontology is far fewer than the total amount of skills located in all the 500 résumés. The skills are also skewed towards skills related to web development. The reason for this limitation is time restrictions. Inserting all the skills would make the tool more usable as a whole, but was deemed too time consuming. Considering that the ontology itself was not optimized and the limitations on time, only the words and skills that were critical to the query testing were inserted into the ontology. Having the words skewed towards one specific field of skills may also create a bias on which sets of résumés are actually retrieved as the context in itself is biased.
One possible solution to the limitations laid on the ontology would be to autogenerate and populate an ontology based on the résumés. This would solve the problem with having to design the ontology and would populate the ontology with minimum effort. Although having that said, auto-generating ontologies is not a trivial task.

7.2.2 Lucene

Another problem that needs to be taken into consideration when working with the Lucene scoring is that the algorithm in general is based on the more times a skill is contained inside a résumé, the higher the score the résumé will get when searching for that skill. Knowing this, a consultant may be able to include skills and experience multiple times in the résumé manipulating the score giving it a higher rank in queries.

7.2.3 Application server

The application server and the hosted user interface used in this thesis are interchangeable to any other languages or frameworks out there. The reason NodeJS and AngularJS are used is because of the authors prior experience with the frameworks. This is to be able to put more focus into the ontology and query augmentation parts of the project rather than struggling with the web application.

7.2.4 Query augmentation

The query augmentation suits well with the intended usage. Having a simplistic solution with the relevant words being suggested keeps the complexity down and is intuitive for users. But this leads to restrictions on what words the query augmentation can suggest. If the query augmentation has gone down one specific field of words, at its current state it will not be able to suggest words that are collected in a different field. This can for example be observed in the query testing results where the last word added to the query "kanban" needed to be added manually. Kanban is a word related to working methodologies and is not directly connected to the web development meaning that the words before had. Considering that "kanban" was the second word in the derived query from the assignment the query augmentation was not able to take this into consideration.

A possible solution to this is to have a similar implementation as SQoogle has, mentioned in. Extending the query augmentation with navigation through the ontology itself would enable the change in context that is needed. The user would then have the ability to visualize the ontology and pick out the words that are necessary. A downside to this is that it adds to complexity. Complexity in the way that the user will have to make more actions to get relevant words as these words could be deep down in an ontology.

The query augmentation limits the SPARQL logic that the user can use for retrieving related words from the ontology and searching the résumés through Solr. The current implementation only supports AND and not other common operators such as OR and NOT. Querying for related words in the ontology can be improved by adding the non implemented operators as it gives flexibility similar to SQoogle's implementation. One could argue that OR operator is more usable than NOT as selecting words that are non-related may not be strict enough. Improving the Solr HTTP request by adding a choice of conditions such as OR and NOT would give it more flexibility. The current implementation enforces that the résumés must contain the words which may be too strict. Adding an OR would lessen this restriction and give the managers more freedom when searching for matching consultants.

7.2.5 Few tests

There are only a few tests that were done during this thesis. The plan was to have iterated user testing and collect more information about what the managers actually add to their
7.3. The work in a wider context

queries. Measurements on how long the query could get, how many times the managers
search before finding a consultant and what relevant words the manager would choose from
the query augmentation are all interesting for the thesis. This is unfortunately not performed
during this thesis as the time allocated for testing was not enough.

7.2.6 Measurement

Precision and recall are the two measurements in focus in the evaluation, also considering
that F-measure consists of both those measurements. These two measurements work better
when the whole set is known, meaning that the correct set of answers and the incorrect set of
answers are always known. In this thesis, the set of résumés is 500 individual résumés and
could be considered an amount that could be divided manually into the correct and incorrect
sets. Although saying that, if a keyword is contained inside a résumé does not mean that the
résumé should be in the correct set. As it was discussed in 2.2.1 the context could be incorrect
making it necessary to analyze the document in a thorough manner to determine which set it
should be in.

7.2.7 Sources and references

Many of the references are to sources that are not from pure scientific literature. The non
scientific literature mostly consists of web-pages to open source projects such as the Apache
frameworks and W3C recommendations. The reason for this is that most of the tools used in
this thesis are of the newer kind. Describing the features that the tools give is better done with
the source itself whether it’s a website or a blog is better than referencing outdated scientific
papers.

A similar case to the one above can be said about some of the sources that actually are
scientific papers, could be considered old. By old, it is old in the world of Internet where even
articles that are a couple of months old could be considered old. Even though discussions
and theories related to the methods used in semantic web have been in circuit quite early, in
our case the oldest resource by Zloof et al in 1975 [33], the web has not since quite recently
adopted the new web standards given by W3C. This can be seen in the lack of resources
between the years 2000-2008. A possible reason for this is that W3C did not present its new
recommendations until November 2008 [32].

7.3 The work in a wider context

Possible improvement to the ontology is to analyze the résumés. As the résumés have a
certain structure it is possible to derive conclusions based on the content contained in specific
parts of the résumé. Although saying this, the résumés themselves are considered company
property. Analyzing the résumés would be of use for the company but could possibly lead to
unintentional insight of the company if seen by third parties.

Another perhaps ethical aspect is that it would be possible, as mentioned in 7.2.2 to ma-
nipulate the score of one’s own résumés. Knowing how the system works, especially with
Lucene and its scoring algorithm, one could gain an edge over one’s coworkers. If this then
becomes common knowledge, all résumés may be focused on including keywords instead of
appealing and well-written text.
The thesis’ main purpose is to improve the search engine at company N. Even though they already had a search engine based on Lucene, many of the managers at N did not know that it was possible to use when looking for consultants. Those who knew deemed it too primitive and problematic as too many results came back from a query. To help N with this, it is proposed to use query augmentation through ontologies for solving the problem with an extensive result set. A proposition was put out where helping users by adding context to the query, removes semantic ambiguity resulting in better precision when retrieving résumés. Important aspects that need to be taken into consideration when implementing a query augmentation are how will the ontology be used?, how to visualize the query augmentation? and will it actually improve the results?

Engineering of the ontology is widely discussed both in the theory and discussion sections. The two main points the discussion circled around were designing a new ontology and reusing an existing ontology. Even though a seemingly well suited reusable ontology existed in the form of the Human Resource Management Ontology [30], it was quickly realized that the Human Resource Management Ontology was better suited for storing information rather than applying relations between the information. Creating our own ontology was considered a better option and gives more freedom when querying with SPARQL.

Creating an ontology has been researched where different approaches and tools are available. As the tool used to create the ontology, Protégé, was created by Stanford, the ontology guideline written by Natalya F. Noy et al [22] was also the basis for creating our own ontology. Even though simplistic, it was well suited for the task at hand. The ontology was then populated by skill related words from the résumés.

The query augmentation was then implemented with a similar visualization as the one Google has on their search site. But instead of suggesting the whole sentences based on old queries, the query augmentation suggests related words to the ones already given.

Regarding hosting and querying the résumés, most of the work was already done by the company. Indexing through Lucene and hosting through Apache Solr greatly helps and simplifies querying the résumés through Solr’s RESTful api. Although much of the résumé indexing and hosting were already implemented, Lucene’s and Solr’s functions were thoroughly researched to help understand the systems.

This was then all connected through different web servers. Hosting the résumés and querying was enabled by Apache Solr. Hosting and querying the ontology was enabled
8.1 Solr and Lucene

Apache Solr and Lucene are powerful tools that help with searching the résumés. Even by itself, it’s possible to narrow down the results if the queries are well defined and specific. Having Apache Solr’s ability to make requests through a RESTful api simplifies the process a lot as any system can through HTTP communicate with the Solr Lucene system.

8.2 Protégé

Protégé is another powerful tool used in the thesis. Protégé simplifies the work of creating ontology considerably through great visualisation of the ontology tree structure and helpers that lessens the complexity that can come with creating ontologies.

8.3 Jena Fuseki

The Apache Jena Fuseki, like Solr, helps with the communication between other systems and the ontology. The tool itself is a great tool for hosting the ontology and its possibilities to parse SPARQL enables advanced queries to the ontology.

8.4 Application

The application itself with a NodeJS server and AngularJS user interface, proved to be quite sufficient for the task at hand. Implementing the query augmentation in AngularJS was not trivial but the end result is pleasing.

8.5 Query augmentation and ontology

By increasingly adding related words to the search query, it was possible to improve the context and improve the results returned from the query. A simple "suggestion"-like query augmentation is deemed sufficient enough to have users add related words. Having a ontology hosting the related words with corresponding relation is a feasible solution. Although possible, ontologies lead to its own problems such as the need to explicitly define and insert the words and relations manually. The current implementation requires a human factor that overlooks the ontology where both consistency and maintenance needs to be checked regularly.

8.6 Further work

Further improvements are discussed in the discussion chapter. Extending the query augmentations visualization would perhaps solve the inflexibility of which context the suggested words are related to similar to SQoogle’s implementation.

The ontology itself could also be redesigned to better suite the résumés. The simplistic approach works well here but may need tweaks to became scalable and work with larger data sets.
One of the limitations on this thesis is that auto-generated ontologies would not be researched in any deeper manner. While doing research connected to the ontology, it quickly became apparent that a lot of manual work would be needed to design, create and update the ontology. Not only is it time consuming but also is error prone due to human errors. Auto-generated ontologies may be the solution to this.

Adding the additional conditions to the queries is also an important enhancement that needs to be taken into consideration. The current AND-only may give résumés with great precision, but may be too strict resulting in relevant résumés being neglected.
Bibliography


Introduktion

Den här intervjun består av två delar där första delen kommer innehålla frågor angående hur processen att hitta en konsult för ett uppdrag går till och om konsulternas CV:n. I andra delen har vi 3 scenarion där vi vill se hur du skulle söka efter ett konsult CV baserat på den information du får.

Vi kommer att spela in intervjun om det är ok för dig? Och vi beräknar att intervjun kommer ta mellan 20-30 minuter.

Intervjufrågor

1. När du ska leta upp en konsult till en förfrågan, hur gör du då?
2. Använder du SearchLight?

Om de använder searchLight:

3. Hur ofta händer det att du hittar flera potentiella CV:n?
4. Om du skulle jämföra hur lämpad en konsult är för ett projekt, baserat på deras CV, vad skulle du då titta på?
5. Händer det att du inte hittar någon konsult efter en första sökning?
   a. Vad gör du i de lägena?
6. Hur väljer du dina sökord?
   a. Förkortningar?
   b. Synonymer
7. I ett scenario där det hade funnits 50 konsulter lediga, tror du att din metod hade varit annorlunda?
   a. I så fall, på vilket sätt?

Om de inte använder SearchLight

Fråga 3 & 6

1. Om du skulle jämföra hur lämpad en konsult är för ett projekt, baserat på deras CV, vad skulle du då titta på?
2. I ett scenario där det hade funnits 50 konsulter lediga, tror du att din metod hade varit annorlunda?
   a. I så fall, på vilket sätt?
Competence Library

Test 2. Jämförelse mellan användandet av competence library och nuvarande arbetssätt

*Required

Beskrivelse av verktyget

1. Öppna verktyget genom att gå in på 10.1.25.154:3000 i en webbläsare.
2. Klicka på LOGIN-knappen för att börja använda verktyget. (OBS. Ibland är inloggen väldigt långsam. Uppdatera sidan om detta händer)
3. Efter inloggning hämtas net-ca. (Detta kan också vara långsam ibland. Uppdatera sidan i det fallet)

Sökning
1. Lägg till ord i sökfältet genom att skriva in ett ord OCH trycka ENTER. Därefter går det att söka genom att klicka "searchlight"
2. Påbörja användning av sökhjälpomen genom att trycka enter efter att ett ord läggs till i sökfältet. (I beta-fas)
3. Resultat visas sedan i den högra tabellen.

Preview och nerladdning
1. Alla .docx kan preview:as via "Preview" - knappen. (Observera att det kan ta cirka 5 sekund för den att ladda)
2. Alla dokument kan laddas ner via "Download"

Instruktioner och uppdrag

Hitta konsulter som skulle passa följande beskrivning. Observera att du inte får söka på personer du känner eller vet skulle passa bra till uppdraget! Vi söker ingen specifik person utan lista de personer du får!

Använd följande metoder:

Metod 1
Använd sökverktyget som finns på din dator och sök genom CV-mappen

Metod 2
Sök via detta verktyg

Uppdraget:

1. Vilka konsulter hittade du via metod 1? *
   (Skriv även frågan du ställde tex "python mongodb")

2. Vilka konsulter hittade du via metod 2? *
   (Skriv även frågan du ställde tex "python mongodb")

3. Hur tycker du att verktyget competence library fungerar? *

4. Vad tycker du kan förbättras?
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsmann i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Huy Tran