Implementing Semantic Search to a Case Management System

Master of Science in Technology Thesis
University of Turku
Department of Computing
Software Engineering
2022
Janne Marjalaakso

UNIVERSITY OF TURKU
Department of Computing

Janne Marjalaakso: Implementing Semantic Search to a Case Management System

Master of Science in Technology Thesis, 84 p., 10 app. p.
Software Engineering
November 2022

The amount of information in today's information society is immense, which creates a need for intuitive and effective search functionalities and applications. In addition to openly available search applications, organizations need internal search functionalities for optimizing their information management. This thesis provides an implementation suggestion for JoutseNet semantic search application. JoutseNet is a case management system used by the authorities and the employees of the city of Turku.

Thesis begins by introducing some relevant fundamentals of natural language processing and search engines. Literature review is utilized to find semantic search implementation methods from previous research papers. Case JoutseNet is introduced with some background information on the case management process and with a brief user research and examination on the current state of the system. Learnings from the fundamental guidelines and conducted research are combined to implement the search application. After the implementation documentation, guidelines for optimizing and testing the application are given.

The value and performance of the implementation is yet to be determined because the production data of the JoutseNet system could not be used for research purposes. A comprehensive suggestion is provided, but further research and development is still needed before delivering it to the production environment.

Keywords: natural language processing, semantic search, Elasticsearch, case management

Contents

List of Figures

1 Introduction

The amount of information grows constantly in today's information society. Web search engines and various platforms providing a search functionality are quite sophisticated systems. Especially Google Search is familiar to practically everyone who uses internet. On top of the basic Google Search, Google has developed optimized search functionalities at least for images, videos, shopping, books, travelling, finance, and academic publications. All these functionalities are quite easy to use even for a new user, which generates high expectations for all other search applications and functionalities.

In addition to Google Search and other openly available search applications, search engines are also needed for finding information from organizational systems. It is highly probable that some ready-made solution is not enough for an organizational search. An internal search application must be tailored for the business and the users' needs. Having optimized search functionalities for information and knowledge management can have a big impact on the overall performance of an organization if the frequency of utilizing those functionalities by the personnel is high.

City of Turku has showed interest in implementing a semantic search functionality for their case management system JoutseNet. The first version of JoutseNet was created in the late 1990s and since then it has been used to store municipal decisions, meeting transcripts and other important documents created by the governmental bodies and the employees of the city. The system is used by various users every day and the number of documents grows constantly. There are search functionalities in the current version of the system, but they are not on par with today's search application standards. The current search capabilities can be very cumbersome and retrieving the desired search results can be very difficult for an average user.

The goal of this thesis is to design and create a semantic search implementation for the JoutseNet system. Before discussing Case JoutseNet, the practical part of the thesis, some fundamentals and known characteristics of natural language processing (NLP), machine learning (ML), search engines and semantics are reviewed in the first chapters. The relevant theories of NLP are explained in Chapter 2 and the significance of ML in the development of NLP is also discussed briefly. Chapter 3 briefly discusses the history of search engines in a general level and presents a concise theory on the creation of a search engine application. In Chapter 4, semantics and semantic search are analysed as concepts and a literature review on semantic search is conducted to gain an understanding on previously utilized methods and techniques for implementing semantic search.

Chapter 5 starts the practical part of the thesis by introducing some background information about the current state of JoutseNet – What is it and why a semantic search functionality is needed? Then, an implementation suggestion for the semantic search application is given in Chapter 6 with an appropriate level of detail so that future developers and users of such application can gain an adequate understanding on the process and used tools. Lastly, Chapter 7 discusses the next steps for Case JoutseNet and some guidelines for testing and optimizing the implementation to support future development of the implementation suggestion.

Chapter 8 concludes the thesis by discussing the results of the practical part and reflecting them to the background theory. Concise answers for the research questions are provided and some thoughts for future research and development are shared. The research questions for this thesis are

RQ1: What kind of methods and techniques are used in previous use cases to enable semantic search functionalities?

RQ2: What are the requirements for a JoutseNet semantic search application?

RQ3: How to create a semantic search functionality, which provides relevant search results for the users of JoutseNet-system?

2 Natural Language Processing Fundamentals

Humans use natural languages, such as English and Finnish, for communication purposes. Natural Language Processing (NLP) is a subfield of artificial intelligence, which studies these languages from computational perspective and, more specifically, the interactions between computers and natural languages. Computer databases are converted into readable language and human language is converted into some formal representation so that computers would be able to process it. Many NLP applications utilize both of these processes for providing a complete service for the user. For example, some system takes a user input, transforms it into machine format, processes the input in machine format and then gives an output, which is first created in a machine format and then transformed to human language so that the user can understand it. [1] The next section introduces some of the most important events of NLP development from the past seven decades.

2.1 Brief Definition and History of Natural Language Processing

The history of NLP started already in the 1950s. It was the branch of science that brought artificial intelligence and linguistics together. The first attempts of utilizing

NLP included a Russian-to-English machine translation. However, especially homographs and metaphors caused difficulties for the translation. For example, with a very simplistic word-for-word approach, sentence "the spirit is willing, but the flesh is weak" was translated to "the vodka is agreeable, but the meat is spoiled". In 1963 John Backus and Peter Naur created the Backus-Naur form (BNF), which is used to specify context-free grammars (CFG). It is generally used to represent programming languages, file formats and protocols and CFGs are also used to model structures of natural languages. BNF specification collectively validates the specified entity with a set of derivation rules. [2]

In 1970s, programming language implementation was simplified by utilizing grammars with lexical-analyser (lexer) generators and parser generators. A lexer takes text and transforms it into tokens and parser validates the generated token sequence. The lexing and parsing decisions are determined by the code and lookup tables, which are based on the predefined regular-expressions and BNF specifications. The programming language Prolog was published in 1972 and it was intended to use for NLP applications with its syntax, which is designed to be well-suited for writing grammars. Originally, NLP was not closely associated with text information retrieval, but during the several decades of development, those two fields have come closer to each other. Today, NLP diversely utilizes knowledge from various fields, which makes it challenging to keep one's knowledge up to date when researching and developing NLP applications. [2]

From the history of NLP, it can be seen that there have been several attempts of defining languages and grammars with strict rules. In most cases, these definitions have not been suitable for natural languages because of their large size, unrestrictive nature and ambiguity. The research has still been beneficial for computer science since it has contributed to the development of programming languages as a positive side effect. Standard parsing and formalized approaches have two main problems when utilizing them with NLP: 1) It is impractical to hand-craft rules for a system that is capable of extracting meaning from text, which is the main objective of NLP and 2) ungrammatical spoken language is natural for a human, but it is very difficult to write rules for such communication. [2] The first problem is about semantic characteristics of language. The topic of semantics will be further explored in Chapter 4 of this thesis.

According to Kumar, the first attempts of NLP in the 1940s and 1950s are parts of the so-called first era of NLP. The second era from 1950s to early 1970s added natural language systems to be a part of artificial intelligence (AI) research and started the utilization of beforementioned parsers. From early 1970s to early 1990s the third era contributed heavily to language processing. Discourse models and semantics were focused on by utilizing scripts, plans, goals and human organization. These building blocks are referred as human conceptual knowledge and they were used to create programs capable of understanding natural language. [1] The LUNAR question answering system was first demonstrated at a lunar science convention in 1971 and it was one of the first systems to use predicate logic as a semantic representation by combining natural language and logic-based paradigms. [3] In the 1980s, more accurate evaluation was performed and large bodies of text were used to train algorithms that utilized probabilistic machine-learning methods. [2]

The current fourth era of NLP, which started in the early 1990s, has standardized the data driven and probabilistic approach. Evaluation strategies from speech recognition and information retrieval are utilized in NLP algorithms. The rapid development of hardware and web applications have provided broad opportunities for NLP. For example, smart assistants, translators, spell-checking and many other implementations would not be possible without the extensive development in the field of NLP. Language-based information retrieval and extraction are important topics especially in the context of web, searching and data analysis. [1] However, NLP applications are still far from perfect. It is easy to understand the difficulty of language understanding and semantics when using some translator tool and the tool cannot take the surrounding context into account.

2.2 The Role of Machine Learning in Improving NLP

Machine Learning (ML) and Natural Language Processing are both important subfields of Artificial Intelligence (AI). Furthermore, Deep Learning (DL) is a subfield of ML, which also has gained much popularity during the past decade and has provided intelligent implementations for various domains. ML and DL have a significant role in improving the efficiency and accuracy of NLP. One could also say that NLP gives the means to communicate with the intelligent systems. Thus, all these subfields enhance each other, and it is appropriate to study the most important intersections of ML and NLP to gain a coherent understanding on the topic. The purpose of this section is to give an overview and a starting point on ML methods as part of NLP, not to thoroughly explain some specific ML methods.

ML enables problem-solving without explicitly programming the computer for the task. For many problems, it is impractical to create an explicit algorithm that provides a solution after it has reviewed all possible inputs. ML and DL help the computers to learn from given data and the systems can make their conclusions and decision based on the learnings. Like in learning generally, with ML methods the machines can be prepared for new problems by basing the solution on their past experiences. Especially DL techniques often require quite a lot of computing resources because of its nature on learning from large amounts of data without explicit instructions. Today, the immense amount of available data and powerful hardware enable wide use of intelligent systems. It is relatively effortless to create a

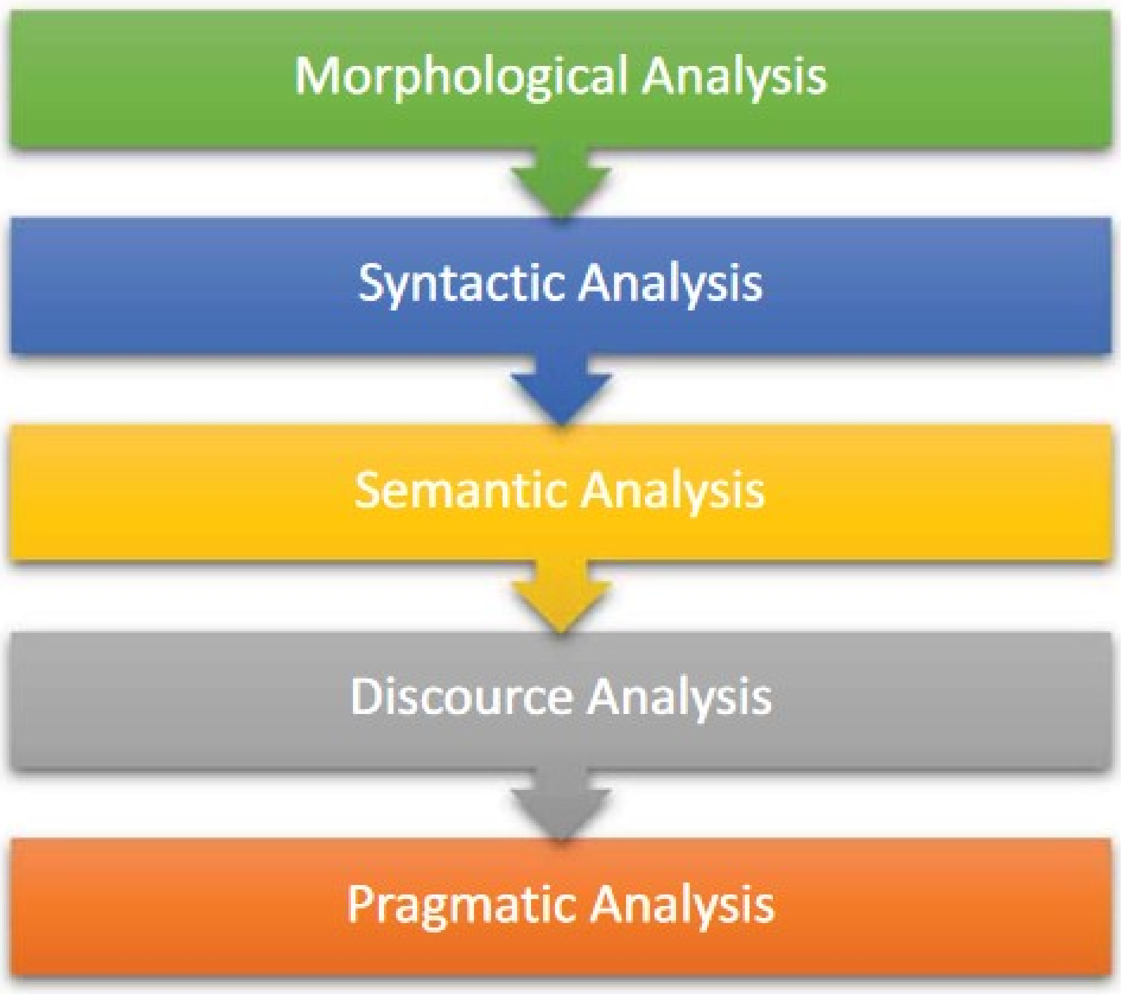


Figure 2.1: The five stages of Natural Language Processing [4]

proof of concept for some ML system with modern tools and frameworks, but deeper implementation still usually requires extensive research and knowledge on the field. [4] Furthermore, one has to keep in mind the uniqueness of a specific use case and carefully craft the system requirements by taking the data, performance, users and other characteristics into account.

NLP is divided into five major stages. These can be seen in Figure 2.1. Nagarhalli et al. have conducted a study on how ML has impacted on each of the stages of NLP. [4] The first stage is morphological analysis, which is about identifying the words and sentences in the given collection of documents. This can be done by tokenizing and stemming the pieces of language. Learning techniques provided by ML can be utilized to improve the performance and accuracy of morphological analysis. For example, sequence labelling, which is a type of pattern recognition, has proven to be a beneficial ML task that improves accuracy and performance of tokenization process. [5]

The second stage, syntactic analysis, includes checking the processed text against the rules of a language. This check ensures that the correct meaning is got from the sentences. However, rules of languages are generally defined explicitly, and syntax can be efficiently checked with rule-based parsers. Thus, there has not been notable usage of ML for the syntactic analysis. [4]

Semantic analysis is the third stage, and it is about understanding the meanings of words and sentences in the given sample. This is a difficult task because, for example, language is ambiguous, and many words have multiple meanings. Context determines the meaning for many words, which is usually natural for a human to understand but can be difficult for a computer to process correctly. The problem of word sense disambiguation (WSD) is considered to be an AI-complete problem, which means that it is at least as hard to solve as the most difficult problems in AI. [6] According to Nagarhalli et al. [4] various ML methods have been used to disambiguate the meaning of words. All types of learning, supervised, unsupervised and even semi supervised, are utilized in different implementations and they have a significant role in solving the problem.

The fourth stage, discourse analysis, goes beyond individual sentences and models language phenomena by uncovering several levels of linguistic structures. This process supports various NLP applications such as machine translation, information extraction and question answering. [7] Let's have the two following sentences as an example text for discourse analysis: "James went to John's shop to check out the new mobile phone. He looked at it for hours." In the second sentence, "He" can refer to "James" or "John" and "it" can refer to "shop" or "mobile phone". The task of determining the references is called reference resolution. Reference resolution is essential when analysing a large text. ML methods such as support vector machines [8] and convolutional neural networks [9] have been used to enhance the performance of discourse analysis.

Sometimes the actual intended meaning of a sentence is different from the written meaning. The intended meaning is picked with pragmatic analysis, which is the fifth and last stage of NLP. [4] For example, sentence "the wedding couple ran around like headless chickens" does not make sense, if it is taken literally as it is written. The intended meaning is that the wedding couple is panicking and senselessly moving from some place to another. Sarcasm detection is one important problem of pragmatic analysis. Sarcasm can quite easily create situations, where the written meaning of text is the complete opposite of the intended meaning. Support vector machine, logistic regression method and random forest algorithm and some other ML approaches have been used for sarcasm detection. [10]

As can be read from the previous chapters, machine learning is an important factor on enhancing the accuracy and performance of many NLP stages and tasks. If some NLP task needs to be improved, it is highly probable that there already is some ML method that can be used as a supporting aid. In the context of this thesis, ML mostly concerns the important topic of semantics and how ML can be possibly utilized to improve the search functionality's ability to return relevant search results.

2.3 Natural Language Processing Terms and Techniques

NLP utilizes various techniques for different purposes. This section introduces some of the most common techniques and terms of NLP so that there is enough background knowledge for Chapters 4-7. Understanding the terms helps to understand the NLP process in practice.

Tokenization

In the context of NLP, Tokenization is a fundamental technique, and it is used for most NLP tasks. Sentences or documents are split into tokens. The tokens can be words or phrases. Tokenization method depends on the processed language. For example, it is trivial to split English sequences by spaces but languages without explicit word boundary markers, such as Chinese and Japanese, require additional methods for word segmentation. Some stop words, such as articles “the”, “a” and “an” can also be removed already as part of the tokenization. [11]

Stop Words Removal

Stop words are common words, which provide very little useful information. They are excluded from further processing of the documents. Some stop words are domain specific. This means that different domains, such as computer domain and medical domain have different stop words. List of domain specific stop words can be created based on the specific domain knowledge. [12]

Stemming and Lemmatization

Inflectional and derivationally related forms of words are reduced with stemming and lemmatization. For example, “speaks” and “speaking” are derived from the same root form “speak”. Furthermore, semantically related forms of a word, such as “history” and “historic”, are reduced to the same root form. [12]

Upper Case and Punctuation Removal

With upper case removal, all processed text is transformed into lower case. This procedure also reduces unnecessary variation of words. For example, “Natural” and “natural” are the same word when all the text is in lower case. Removing punctuation is also beneficial because they increase the size of the data and are not very useful in text analysis. [12]

Part of Speech Tagging and Parsing

Lexical and syntactic information are analysed with Part of Speech (POS) tagging and parsing. Each token is given a POS tag, which tells if the word is a noun, an adjective, a verb or some other part of speech. Some use cases can highly benefit from this information. For example, in opinion surveys, adjectives are usually opinion words and nouns are the targets of opinions. Parsing brings structure to the information and provides the syntactic side of the analysing with a grammatical tree. [11]

N-grams

N-grams are one of the most fundamental text features and it is commonly used in NLP. An n-gram is a sequence of occurring n items in a sample of text or speech. The items can be letters, syllables, words or even phonemes. Window size defines the sequence length. Window size of one is a unigram, two is a bigram, three is a trigram etc. For example, unigrams of “natural”, “language” and “processing” and bigrams of “natural language” and “language processing” can be picked from the phrase “natural language processing”. [11][12]

Term Frequency - Inverse Documents Frequency (TF-IDF)

TF-IDF expresses how important a word is to a document in a collection. Terms appearing frequently within one document are more important than terms appearing frequently in multiple documents. [13] The value of TF-IDF is calculated with the following equation:

$$TF - IDF = \left(\frac{n_t}{N}\right) \cdot \log\left(\frac{K}{DF_t}\right) \tag{2.1}$$

where:

n_t	= occurrence of term t within a document
N	= number of terms in the document
K	= total number of documents
DF_t	= number of documents containing term t

Named Entity Recognition

Named entity recognition (NER) groups words and phrases to pre-defined categories. For example, the categories could be people, organizations, time, and locations. In sentence “Janne has lived in Turku since 2015”, “Janne” is a person, “Turku” is a location and “2015” is put in the time category. [2]

2.4 BERT

In 2018 Google published a language representation model called BERT: Bidirectional Encoder Representations from Transformers. According to the original research paper of BERT, the model is “conceptually simple and empirically powerful”. Just one additional output layer is enough to create state-of-the-art models for various NLP tasks. [14] BERT has been a significant actor in rapidly advancing NLP generally during the past few years [15] and it is even considered to be the new industry standard for word processing. [16] Thus, it is relevant to introduce BERT when composing a brief overview of NLP. Furthermore, BERT is a noteworthy candidate for aiding a semantic search functionality, which ties it naturally as a part of this thesis. [17]

BERT is a pre-trained language model, which means that it has been trained with a very large corpus and it can be further finetuned with some specific collection of texts for some specific purpose. [18] For BERT, the pre-training was performed with BooksCorpus (800 million words) and English Wikipedia (2,500 million words). Lists, tables and headers were ignored from the Wikipedia material, which left only the text passages to be extracted for the training. [14] BERT uses transformers to teach how to represent text. Generally, transformers are a type of neural network that is used for ML problems where inputs and outputs are sequences. [16] They were first introduced by Google in 2017 and they enable bidirectionality, which is a core characteristic of the BERT model. Bidirectionality means that the model is able to read text from both directions, left-to-right and right-to-left, at once. Before this feature, language models could only read text sequentially in one direction by using recurrent neural networks (RNN) and convolutional neural networks (CNN) for NLP tasks. With the new capability, BERT is trained on two NLP tasks, which are Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). MLM hides a word in a sentence and then the model predicts the hidden word based on the context. NSP predicts the relationship of two sentences: They either have a sequential connection or their relationship is random. With the ability of processing data in any order, transformers can train on larger amounts of data and models like BERT are possible with their substantial training data. [19]

Already in the research stages, BERT achieved impressive results in many NLP tasks. Previous popular language models, such as word2vec and GloVe, were not capable of such advanced interpretation of context and polysemous words. Even ambiguity can be addressed with BERT, which brings it closer to human-like common sense. Again, this is enabled by transformers, which process each word in a sentence in relation to all other words in that sentence and builds context with the observed relations. Google emphasizes that it is not recommended to optimize one's web content for BERT: The goal is to have a natural search experience with natural content. [19] BERT is also constantly being developed to work better with more languages. Already in 2020, multilingual BERT (mBERT) achieved good performance on multiple NLP tasks and was applied to over 104 languages. [20]

One important aspect of BERT from the developers' perspective is that it is open source. Anyone can use the model and train it to some specific need. Many organizations and research groups are fine-tuning the model to specialize it and to optimize its efficiency. For example, there is docBERT, which is fine-tuned for document classification and SciBERT, which is fine-tuned for scientific text. BERT has already had a large impact on various NLP tasks and the impact is expected to grow even larger. Voice search and text-based search are both benefitting highly from this new approach. BERT even has potential for providing an exceptional boost for AI systems in general with the efficient ability of training the machines to "understand". [19]

2.5 FinBERT

As mentioned in the previous section, there are many projects, that have fine-tuned BERT for a specific purpose. One of these projects has produced FinBERT, which is a BERT model for Finnish published by Turku NLP group in 2019. [21] Turku NLP group is a group of researchers at the University of Turku and the UTU graduate school. According to their website, they mainly focus on researching various aspects of NLP, language technology and digital linguistics. Their applications have mainly focused on the domain of biological, biomedical, and clinical text. Additionally,

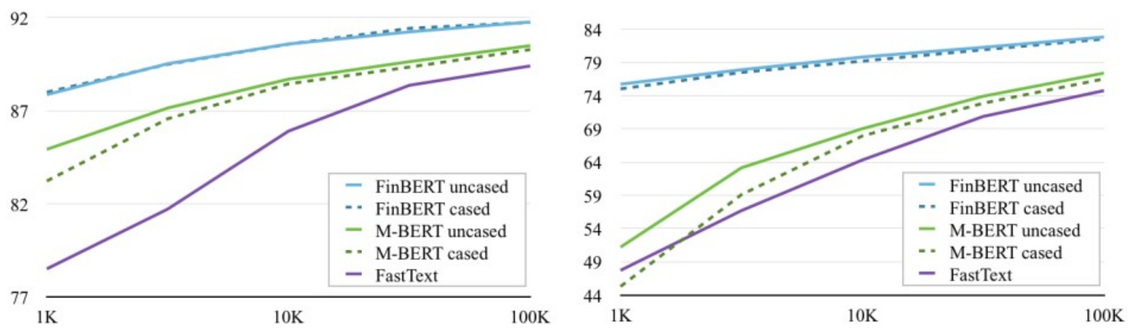


Figure 2.2: Accuracy of text classification with training data sizes from 1K to 100K. Yle news on left and Ylilauta online discussions on right.[21]

methods and resources for syntactic and semantic analysis of Finnish and modelling of web-based language use is researched. [22]

Multilingual BERT models have managed to handle high number of languages impressively well. However, monolingual models can still remarkably outperform the multilingual ones. By developing FinBERT, the research group aimed to provide a model, which outperforms the multilingual options. Finnish news, online discussions and an internet crawl were used for pretraining the model. The news were extracted from the corpora of Yle (national public broadcasting company) and STT (Finnish News Agency). The Yle news were from years 2011-2018 and the STT news were from 1992-2018. Altogether, the news sources resulted in 4M documents, 68M sentences and 0.9B tokens. The source for the online discussions were the Suomi24 corpus, which contained all Suomi24 posts from years 2001-2017 and it provided 4.5B tokens. The internet crawl data was obtained from crawl of Finnish internet that was performed during years 2014-2016 and from Common Crawl Project [21], which is an open repository of web crawl data. [23] These two sources resulted in 8.1B tokens. Thus, before cleaning up the data, there were a total of 13.5B tokens. The cleaning up and filtering included removing headers, tags, documents with low average sentence length, duplicates, and other redundant pieces of data. After that, the number of tokens were 3.3B, which still was around 30 times the size of the Finnish Wikipedia that was included in the training of multilingual BERT. [21]

Multiple NLP tasks, such as part of speech tagging, named entity recognition and text classification were used to evaluate the model and to compare FinBERT to multilingual BERT. For example, the text classification had over 20% higher performance with FinBERT when using a small set of training data. The research shows that for Finnish, which is a lower-resourced language, it is possible to create a model that performs better than the multilingual BERT. [21] BERT and multilingual BERT are good options, but it is advisable to look for other possibilities also, when one is designing an NLP system for some specific purpose.

3 Search Engine Fundamentals

3.1 Brief History of Search Engines

Search engines have become a common part of everyday lives. They can be used to find information about almost any existing entity. For example, one could search the location of a restaurant or the birthday of some celebrity. Search engines, mainly Google Scholar, are heavily utilized to find source material for this thesis. According to Seymour et al. web search engine can be defined as follows: “Web Search Engine is a software program that searches the Internet (bunch of websites) based on the words that you designate as search terms (query words). Search engines look through their own databases of information in order to find what it is that you are looking for. Web Search Engines are a good example for massively sized Information Retrieval Systems.” [24]

Google search is clearly the most well-known search engine, which has led to the fact that “googling” is used as a synonym for searching information on the web. However, there are many other search engine alternatives and Google has not been the first or the only actor in the field at any point. The first software for searching on the Internet was created in 1990 by computer science students Alan Emtage, Bill Heelan and J. Peter Deutsch at McGill University in Montreal. The searching tool was called Archie and it created a searchable database of file names by downloading the directory listings of all files stored on public anonymous FTP sites. [24] The last known Archie search tool is offered by a Polish university. [25]

The number of websites started to grow in the mid-to-late 1990's, which resulted to increasing development of search engines. WebCrawler was published in 1994 and it was one of the first full text crawler-based search engines. With WebCrawler it was possible to search for any word in any webpage and it steered the direction towards the searching standards we have today. [24]

In the late 1990s, there was a surge of investing in internet applications. Search engines were a significant part of this surge and many companies managed to receive record gains with their initial public offerings. Some companies started with a public search engine but have taken down it since and offer only enterprise editions of their services. The dot-com bubble also delivered major investments to some search engine projects, which turned out to be completely fruitless. [24]

In the turn of the millennium, Google's search engine started to gain its popularity. Google introduced PageRank, which helped the engine to achieve better search results when compared to competitors. PageRank is an iterative algorithm, which uses the number and PageRank of other web pages that link there to rank all searchable pages. The ranking is based on the fact that desirable pages are linked to more than others and it managed to achieve a good correlation with the human concept of importance. Previously used keyword-based methods mainly used the frequency or association of the search terms to rank the search results. Google search was also one of the first search engines to have a very minimalist interface, which gave an edge over the web portal implementations of the competitors. The most notable competitors for Google Search have been Yahoo! Search and Microsoft Bing (formerly MSN Search). [24] During recent years, DuckDuckGo has become one of the most significant competitors for Google. DuckDuckGo claim that they do not collect or share any of the user's personal information, which is an advantage in today's world, where more and more users are aware of the constantly encountered privacy issues. [26]

3.2 How to Create a Search Engine Application for an Organization

Public search engine applications such as Google Search, Microsoft Bing and DuckDuckGo are generally good tools for information retrieval and their capabilities are quite vast, but companies still need to create their own solutions for searching information from their own databases and to optimize the workflow of their employees. The amount of stored information grows constantly, and organizations have to develop and maintain search functionalities, which enable efficient utilization of the stored information. This section will go through the most important parts and theories of search engine software development.

3.2.1 Define the Requirements

Creating a search engine application is not a straightforward task. Like most software development projects, search engine development also starts with defining the requirements. Every search engine created for a specific purpose is different and one is required to ask the right questions so that the users can find relevant results: What kind of content is searched? Who are the users? What kind of queries are used and how is the vocabulary? How much data is there? How large will the index be? How many searches the software should be able to handle? How often should the index be updated? Which features are relevant and should be included? These questions are not enough to tell the whole truth about the requirements, but they provide a good starting point for the development process. Additionally, the organization's ultimate intent for the search functionality should be evaluated: Is it important to gain profit or page views with the solution or is it only for finding

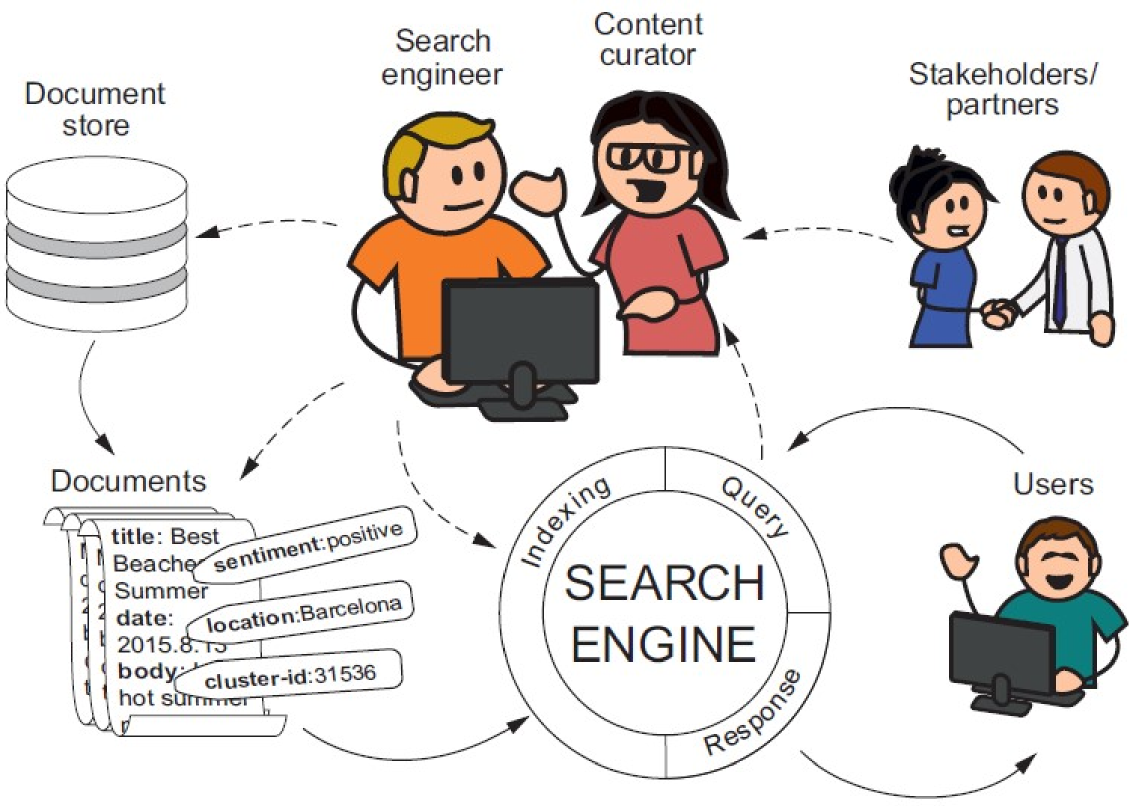


Figure 3.1: The relevance/search engineer implements the search application and works closely with a cross-functional team [27]

documents from an internal database and making the employees life easier? [27]

Relevance is the most important attribute when creating a search engine and it has to be in mind during the whole process, from defining the requirements to deploying and maintaining the software. Relevance may also be the most challenging part of the development process. Doug Turnbull's and John Berryman's book *Relevant Search* (2016) focus on the relevancy problem and tell the reader how it should be approached when building a relevant search experience. Creating an appropriate relevance ranking requires research and effort, which may cause a shift to developing the less problematic parts of the application like performance or user interface. [27]

Relevance and search engine software overall are hard to implement because users are used to easy and intuitive searching and information retrieval. Users expect to find relevant results with low effort and without extensive knowledge about the search functionality. Users get easily frustrated and lost if they have to use an information system, which does not have a search bar that is easy to access and use. Relevance expectations are different with each search application and they should be carefully defined when gathering the requirements. For example, an expert search application created for medical professionals should understand that “heart attack” is the same as “myocardial infarction”. [27]

According to Turnbull and Berryman, “a relevance engineer transforms the search engine into a seemingly smart system that understands the needs of users and the business”. Relevance engineer or search engineer is the technical person, who implements the search application. Despite the title of relevance engineer, other nontechnical people involved in the process usually are the experts when defining relevant results. People who understand the content, business and users are the ones that can define the appropriate search experience with their knowledge and experience. [27]

3.2.2 Select an Engine

The next step is choosing an engine for the application. There are various options to choose from. According to DB-Engines ranking of search engines, Elasticsearch, Splunk and Solr are the most popular enterprise search engines at the time of writing this thesis. [28] However, Splunk is a platform for observing machine-generated data, not a platform suitable for searching documents, which takes it out of the context of this thesis. [29]

Comparing Solr and Elasticsearch

Apache Solr was first released to open source in 2006, when it was established as a subproject of Apache Lucene. In 2021 Solr was established as a separate top level project. [30] During its first years, Solr led the search engine field, and it was the first choice for most enterprises wanting to implement a search functionality for their own needs. In 2010, Elasticsearch was released, which provided a new prominent option when choosing a search engine. It is also built around Apache Lucene, which makes its basis similar to Solr's, but functionalities and development direction in general have defined differing strengths, weaknesses and use cases for these two search engines. Elasticsearch did not dethrone Solr immediately after becoming available, but its modern principles and its capabilities of handling large indices and high query rates certainly did influence the gained traction on the market. [31]

Solr and Elasticsearch are both released under the Apache Software License, but only Solr is genuinely open source. There is not a single company controlling the source code of Solr and anyone can contribute to it. A group of developers, so called committers, can directly write to the source repositories. Based on merit and continued contribution, developers can be elected to be committers. [30] Elasticsearch has been defined as open source for most of its lifetime, but in 2021 Elastic company announced changes to their licenses. They shifted from licensing their source code under the Apache 2.0-license to licensing under the Elastic License and SSPL 1.0 (Server Side Public License). After the changes, their software could not be approved by the OSI (Open Source Initiative) and the company had to leave out "open source" from their products. Now the products are referred to as "Free & Open", with which they emphasize that the product is free to use, the source code is available for everyone, and they maintain an open and collaborative model in GitHub. They promise to remain committed to transparency, collaboration, and community, which are the principles of open source. However, only the employees of Elastic company can become committers. [32]

Documentation does not directly define the quality of some software, but it is an important factor, when developers choose the tools and frameworks they want to use in their projects. Elasticsearch has some advantages over Solr when comparing their documentations. Elasticsearch provides clear structure, examples and configuration instructions on its website. Solr's documentation is a bit more difficult to navigate and finding good technical tutorials and examples require more effort. Some developers consider Solr's documentation to be out-of-date and the coverage of APIs to be too minimal. [33]

For Solr, Apache maintains APIs for Java, Flare, PHP, Python and Perl. The Java one is most up-to-date and well maintained of the APIs. There are other APIs also, but they are maintained by the community. Elastic company directly develops and supports multiple APIs for Elasticsearch: Java, JavaScript, Groovy, .NET, PHP, Perl, Python and Ruby. Some other APIs are also developed by the community. [34]

The configuration of Solr and Elasticsearch indices have some fundamental differences. The default procedure for Solr is to create a schema file, which defines the index structure, fields and types. [31] Solr has a Schemaless mode, but it requires its own configuration. [35] Elasticsearch can be used as schemaless from the start. In practice this means that documents can be sent to a new instance of Elasticsearch without any index schema and the engine tries to guess appropriate field types. This eases the configuration process, but can introduce some inaccuracy, since the guesses are not always right. The schemaless configuration may be a useful feature to have, but defined index structure makes the entity easier to manage in the long run. [31]

Solr and Elasticsearch are both built around Apache Lucene, and they fully take advantage of the near realtime search (NRT), which was added to Lucene's IndexWriter in version 2.9. NRT makes it possible to find documents within milliseconds after they have been updated to the index. [31][36] However, there are a few differences between the engines when comparing their search features. Both are suitable for text search, but Solr has some advantages on the full-text search capabilities and Elasticsearch has its own advantages on the analytic side. [33] For example, Solr has flexible possibilities for using suggestions and spell checkers and a highly configurable highlighting support. Elasticsearch provides a "suggesters" API, which is easy to use, but a ready-made API limits the flexibility. The highlighting of Elasticsearch is also less configurable. [31] Solr is more text-search oriented and Elasticsearch has better capabilities for analytic purposes, such as filtering, grouping and aggregation. [31][33] Elasticsearch focuses a lot on making the analytic queries more efficient and lowering the needed computational power. These optimizations are done also at the Lucene level in addition to the Elasticsearch level. [31]

In terms of performance, there are no big differences between Solr and Elasticsearch. Of course, some use cases require further testing with some particular data, but in most cases the engines have equal performance levels. [31] According to the analysis and comparison study of Akca et al., it is difficult to predict the performance of either search engine before testing them on the particular purpose of work. [37]

Conlusion

Thanks to the popularity of Solr and Elasticsearch, they both have large communities behind them and both engines have an extensive set of features already. They are mature projects that have been the first choices for search application developers for many years already. Solr may be more appealing to developers that value the open-source aspect and want their tools to be truly open source. Solr may also be a beneficial choice, if the receiving company or client has previously already invested a lot of time in Solr or have used Apache products or some other frameworks that are built to work with Solr. Elasticsearch is easier to get started with, easier to scale and has a well-designed user interface and documentation, which makes it a tempting choice, if the developer or company does not want to use much time on configurating and studying the system. [31][34] Blasenak writes that "Solr is like Linux. Elasticsearch is like Windows". This refers to the customizability of Solr and to the easier management and deployment of Elasticsearch. [34] All in all, there are no clear guidelines on choosing between Solr and Elasticsearch and both are definitely good choices for most use cases needing search functionalities.

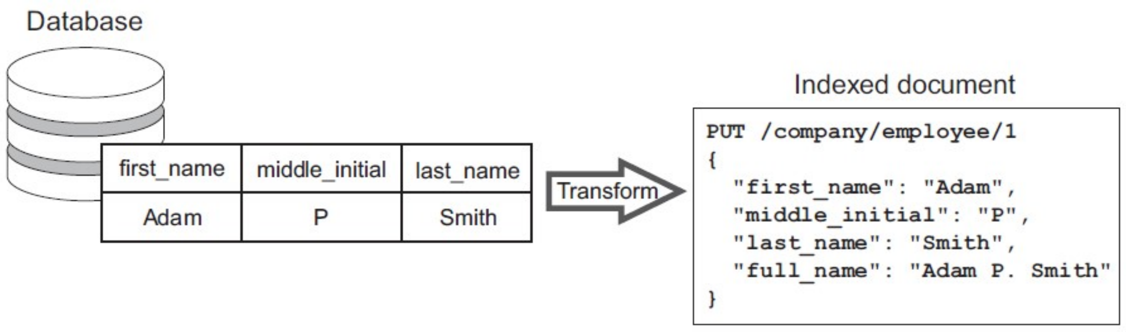
3.2.3 Implement Software Around the Search Engine

Just a search engine is not enough for creating a search application. A search application needs a user interface for it to be usable and approachable by the end users and a backend service for implementing custom business logic. This section will go through some of the most important building blocks for a search engine to gain a high-level understanding on the development process and on the structure of a search engine application.

Define the Index Structure

One of the first steps in configuring a search engine is defining the index structure. Schemaless modes can also be used, but self-defined structure provides higher control and predictability over the used data. For example, differentiating full-text string fields and exact value string fields, using custom date formats and using some specific data types is possible by defining the structure explicitly. It may even be beneficial to index the same field in multiple ways to use it for different purposes. For example, a text field can be indexed as a text field for full-text search and as a keyword field for aggregation. [38]

Most search engines use a data structure called inverted index, which consists of



```
PUT /company/employee/1
{
  "first_name": "Adam",
  "middle_initial": "P",
  "last_name": "Smith",
  "full_name": "Adam P. Smith"
}
```

Figure 3.2: Transforming the database model to an indexed document [27]

two main parts: a term dictionary and a posting list. All terms that appear in the indexed documents are listed and sorted in the term dictionary. The posting list keeps track of the term occurrences in the documents. In other words, all terms are pointed to lists of documents so that it is easy to retrieve the documents containing the queried term. Usually, documents contain several fields, and the terms are first sorted by field and then sorted lexicographically within the fields. It is common for inverted indices to include some further structure and metadata for aiding the search relevancy. For example, doc frequency is the number of documents that include a particular term and term frequency is the count of term occurrences in a particular document. These pieces of information can be enabled or disabled when optimizing a search application. [27]

Search engineers must carefully design the index so that it answers to the needs of the use case. According to Turnbull and Berryman, searching is often made complex because of insufficient signal modelling. Signal is “any component of the relevance-scoring calculation corresponding to meaningful, measurable user or business information”. In other words, it is any information that can be used to rank the search results. In many cases, the indexed data is not originally in an optimal form for the search functionality. For example, in a database a person’s name could be divided to first name, middle name and last name. For searching purposes, it is likely to be more beneficial to index the names as single, combined strings, not as three separate strings. [27]

Implement Backend for the Application

Like generally in software development, it is highly recommended, or even required, to utilize a backend in a web application. In the context of search applications, one of the most important benefits of backend implementation is the possibility to customize the search queries and to use the appropriate attributes for answering the users' needs. More general purposes like central data access, privacy and security are also relatively easy to satisfy and control with centralized backend operations.

To keep the search application up to date, the data has to be regularly updated from the database to the search engine. In some cases, the data can be directly steered to the search engine from the database, but it is likely that some additional code is required for the update. Cyclic updates can be achieved with task queues, such as Python Celery, or messaging libraries, such as ZeroMQ. The data update latency depends on the use case. It may not be appropriate to update the search engine index after every addition to the database. The general guideline is to keep index updates quite rare, which is again a relative definition. Some use cases may require updates every few minutes and for some other use cases updating once per day or week may be enough. [39]

Retrieving the data and defining the index structure are good to have as parts of the backend, but a quite large portion of it is probably reserved for receiving the query, optimizing it and sending it further to the search engine. In this case, optimizing means using the query parameters appropriately according to the specific business logic, which includes targeting the correct document fields, filtering, sorting, and many other possible operations. The goal is to generate a list of results, which are properly scored and sorted. The definition of proper results is based on the expectations of users.

Implement Frontend for the Application

Applications utilizing search functionalities are so common today that every person using computers or mobile phones have some level of expectations on how a search functionality should work and how it should look like. The current industry standards are already so high that even relatively minor problems can cause major frustration. Google and other major actors in the industry have created a mental image of a simplified search bar that is very easy to use, and desired search results are found quickly with very low effort.

According to Martin White, starting with the user interface (UI) may be a good idea when developing an enterprise/organization search application. [40] Defining the required filters, snippets, sorting options and everything else that is needed for the UI helps to form an understanding on the needed backend operations. For example, it may be redundant to implement a certain filtering possibility in the backend, if that functionality is not needed by the users when they are interacting with the frontend via the UI. From the defined requirements the developers can derive, which functionalities and views are really needed for the UI and are not just a waste of screen space.

Especially organizational information collections contain many documents that are almost identical with each other. This is because many of those documents use the same templates, and they are produced as a part of some routine operations. Similar documents complicate the task of showing relevant search results and lifts the importance of relevant filter and sorting options. It is also important to carefully consider the appropriate snippets that are showed along the search results. For some results, it may be essential to show a snippet with highlights, but some other results may not benefit from that, and it is better to show just the higher-level information to make the result view more compact. [40]

According to a survey of Businesswire, 57 percent of employees state that complications in finding the correct information is a major actor for lagging productivity. [41] This emphasizes the fact that the developers need to understand the users during the whole development process. The logic in the backend has to be appropriate for the use case but the UI has to also be convenient and to avoid confusing and frustrating features. When comparing to web search, organizational search only needs to deal with a fairly limited pool of information. Limited environment also enables better awareness about the users and their intents and actions, which makes it possible to design an optimal experience for the specific needs. If the used system requires logging in, it is possible to analyse individual users and to provide pre-filtered search results based on their roles. Differing information needs may require some additional engineering with the search functionality. [42]

Inappropriately handled errors and waiting times can easily cause confusion among the users of the search functionality. The state of search should be always clear to the user. [42] If the performed search takes time, the user should at least be informed about it with a simple but clear message in the UI. If there are no search results at all, it should also be informed to the users. Technical errors should trigger an informative message, which tells enough so that the user can react appropriately and be in contact with right personnel to fix the problem. Too dramatic or technical error messages are not a good idea. They only generate negative experiences and the users may even get scared of the system. [43]

4 Semantic Search Theory and Possibilities

4.1 Definition of Semantics and Semantic Search

Encyclopedia Britannica defines semantics as follows: “The philosophical and scientific study of meaning in natural and artificial languages.” The noun *semantics* is derived from a Greek adjective *sēmantikos* (“significant”), which is a derivative of the verb *sēmainō* (“to mean” or “to signify”). Semantics, semiotics, semology and semasiology are terms that are used interchangeably in literature, but semantics is ultimately used as the name for the study of linguistic meaning. [44] In other words, semantics studies the meaning of words, phrases, sentences, and other linguistic entities, not the meanings of actions or phenomena. [45]

In semantics, meaning is divided into three levels. The levels are expression meaning, utterance meaning and communicative meaning. Expression meaning can be analysed from a phrase when it is studied in isolation without any context. Utterance meaning is the meaning of a phrase in a given context, which gives the utterance fixed reference and truth value. Communicative meaning takes the phrase to a social setting and uses it as a communicative act. Furthermore, expression meaning can be examined under the scope of lexical meanings, grammatical forms and syntactic structure. Semantics is an extensive theory about everything related to linguistic meaning and it includes concepts such as readings, logical properties of sentences, predications in sentences, relation to cognition and language comparison. [45]

Semantics works as a background theory for semantic search. Semantic search can be described as a search with meaning. Fifteen years ago, even the biggest web search engines were mostly lexical, which means that those engines utilized only literal matches of the queries or at most some variant of the typed queries. They were not capable to understand the meaning of the query words and use that information for retrieving relevant results. [46] For example, search query "Port of Turku" could only find results, which include exactly some of the query words, but the search engine could not understand that port is a place for ships or that it is a synonym for harbour or that Turku is a city in southern Finland. A semantic search engine or application is designed to understand those facts and to retrieve results that could be related to the query, even if the results do not include any of the query words. Semantic search accepts a wider variety of queries for finding relevant results. It makes searching easier for the user when it "understands" the semantic aspects of the queries and the indexed documents. The "understanding" behaviour can be achieved with various natural language processing tasks, such as named-entity recognition, sentence parsing and word vectors. [46]

Semantic search does not have a clear and unambiguous definition. Researchers from different fields work on a large set of distinct issues related to semantic search. It brings various benefits and means different things to different people. This leads to the fact that many fields of study or work have their own definitions of semantic search with their own specific use cases. [46]

4.2 Literature Review on Semantic Search

This section reviews some previous studies and use cases of semantic search. The gained knowledge is used for answering the first research question RQ1: What kind of methods and techniques are used in previous use cases to enable semantic search functionalities? The purpose of RQ1 and the literature review is to give ideas for implementing the semantic search for the JoutseNet system. The implementation of the case specific semantic search functionality is documented in Chapter 6.

4.2.1 Method of the Literature Review

A Systematic Literature Review (SLR) is followed to review groups of studies for this thesis. With SLR, all studies of a field are captured, which can improve a traditional review because it also includes many lesser-known publications, and the studies are not biased towards the research area or the interest of the researcher. [47] The captured studies are used to answer a research question. For this thesis, the method has to be slightly modified and limited to avoid excessive workload.

The first step in the literature review is to find as many relevant semantic search papers as possible. The search will be done with keyword combinations that are introduced later in this section. There should be at least two applicable papers to avoid too limited scope and bias towards some specific idea or result. If there are more than 10 relevant papers found, then the first 10 papers are chosen for further examination. The search process ends when at least 10 relevant papers are found, or all predefined keywords are used. The selected papers must, at least on a high level, describe the implementation methods for adding semantic characteristics to some search engine, which preferably has a specific use case with relatively limited environment. Limited and organizational search systems are preferred in this review, but web search studies can also be utilized, if the semantic functionalities seem relevant and could be utilized in case JoutseNet. Text retrieval is preferred over any other type of information retrieval because case JoutseNet is about finding documents, which are only in textual format.

Google Scholar is used widely for finding sources for this thesis. It will also be utilized to find papers for the literature review. Searching, reading and selecting the papers are all done manually. Each search query is a combination of keywords “semantic”, “elasticsearch”, "document retrieval", "information retrieval", “implementation” and “case study”. Elasticsearch has already been selected to be the search engine for case JoutseNet, so it is justified to use it as a keyword, which provides relevant search results in the context of this thesis. It is expected that the keyword filters out the results concerning web search implementatios as Elasticsearch is primarily an enterprise search engine. After a keyword combination is used as a query, the search results are examined within the first five pages of results. The searching process is started with the initial search and if the initial search does not provide enough relevant results, the searching will continue with another combination of keywords, which most likely will retrieve more results. Keyword combinations are used in the following order:

1. semantic elasticsearch document retrieval implementation case study
2. semantic elasticsearch document retrieval case study
3. semantic elasticsearch document retrieval implementation
4. semantic elasticsearch document retrieval
5. semantic elasticsearch information retrieval
6. semantic elasticsearch

Titles and abstracts are important factors on defining the relevancy of the studies. If the title and abstract of some study indicate the study could provide valuable and relevant insight on the topic, the content can also be scanned. A study can be added to a list for further examination if the content proves to be useful. The list of selected studies will be further examined to find answers for RQ1. The most important parts of the studies are the main characteristics of the use cases, the implementation method of the semantic comparison and the results.

4.2.2 Criteria for Selecting the Papers

There has to be some principles for choosing the papers in the literature review. The following criteria is used to choose the papers for further examination:

- Paper is written in English
- Paper is freely available without special permission or fee
- Paper is found within first 5 pages of search results
- Paper discusses semantic search in the context of textual information retrieval
- Paper describes the method that enables the semantic characteristics
- Paper is published between 2012-2022

4.2.3 Selected Papers

P1. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search [48] keyword: "semantic elasticsearch document retrieval implementation case study"

P2. Geospatial and Semantic Mapping Platform for Massive COVID-19 Scientific Publication Search [49] keyword: "semantic elasticsearch document retrieval implementation case study"

P3. **Semantic Vector Encoding and Similarity Search Using Fulltext Search Engines** [50] keyword: "semantic elasticsearch document retrieval implementation case study"

P4. **Information retrieval and analysis for a Modern organization** [51] keyword: "semantic elasticsearch document retrieval implementation case study"

P5. **Keyphrase Extraction Model: A New Design and Application on Tourism Information** [52] keyword: "semantic elasticsearch document retrieval implementation case study"

P6. **Enhancing Clinical Information Retrieval through Context-Aware Queries and Indices** [53] keyword: "semantic elasticsearch document retrieval implementation case study"

P7. **Semantic Search System using Word Embeddings for query expansion** [54] keyword: "semantic elasticsearch document retrieval implementation case study"

P8. **Natural language processing methods for knowledge management — Applying document clustering for fast search and grouping of engineering documents** [55] keyword: "semantic elasticsearch document retrieval implementation case study"

P9. **Strings and Things: A Semantic Search Engine for news quotes using Named Entity Recognition** [56] keyword: "semantic elasticsearch document retrieval case study"

P10. **Corpus domain effects on distributional semantic modelling of medical terms** [57] keyword: "semantic elasticsearch document retrieval case study"

The first 8 papers were found with the initial set of keywords: "semantic elasticsearch document retrieval implementation case study". These keywords provided 4720 results. The last 2 papers were found with the second set of keywords "semantic elasticsearch document retrieval case study", which provided 4900 results. Almost all selected papers were found with the initial search. There were some search results that initially seemed noteworthy but turned out to be irrelevant with further examination: Some cases were more about the data structure and the data flow than the search functionality and the implementation of the search functionality was not the main topic of the paper. Some papers described the optimization of a search engine for some specific use case, but they did not utilize any techniques for extracting semantic characteristics from the used data. There were also some cases regarding logging and big data analytics, which were left out of further examination.

4.2.4 Brief Description of Selected Papers

The first selected paper *CodeSearchNet Challenge: Evaluating the State of Semantic Code Search* [48] describes the challenge of using natural language for searching some specific snippets of source code. The authors of the paper hope that the introduced challenge inspires further research on the topic and even that competition and leaderboard are hosted to track the challenge. The codesearch use case is very broad with a large corpus of many programming languages. The implementation examples make the paper relevant for this literature review: Some baseline codesearch models are introduced with tokenization, sequence encoding and code/query embedding. The semantic similarity is based on the angular distance between code embeddings and query embeddings.

Geospatial and Semantic Mapping Platform for Massive COVID-19 Scientific Publication Search [49] focus on delivering a platform, which helps in navigating the large pool of COVID-19 research that has emerged quickly and is difficult to keep up with. The visualization and geospatial map are important parts of the platform, but semantic mapping is also there to equally provide value. The workflow of the platform development includes data cleaning, embedding, dimension reduction and visualization. The users of the platform can select from three supported embedding methods: bag of words, GLOVE word vectors and BERT embedding.

In paper *Semantic Vector Encoding and Similarity Search Using Fulltext Search Engines* [50] the authors demonstrate a method that enables vector similarity searching for standard fulltext engines such as Elasticsearch. The solution is applied to a dense vector representation of English Wikipedia. The vector encoding is done with a specific process, which includes rounding and interval encoding. The vectors are trimmed and ordered with high-pass filtering techniques.

Information retrieval and analysis for a Modern organization [51] introduces a method for implementing a search engine system for a large multifaceted organization with large volumes of data. The amount of handled data is immense, and it constantly grows even larger, which brings more challenge to unifying the data systems. The paper lists some important requirements for a search engine of a large organization. For example, it is important to support a large number of formats, document metadata (such as authors and auditors) has to be preserved, content has to be indexed and it is valuable to have a context search to enhance the information retrieval. In this case, “context search” is the attribute that brings semantic characteristics to the system. It is implemented with Latent Dirichlet allocation (LDA), which is a probabilistic topic model. Specific weights are added to the indexed documents with the LDA model and then contextually related documents can be grouped based on those added weights.

Keyphrase Extraction Model: A New Design and Application on Tourism Information [52] describes how keyphrase extraction can be done to enhance searching techniques in tourism. Tripadvisor.com, Agoda.com and vietnam-guide.com are used as sources for the tourism data. BERT and Glove embedding, Bidirectional Long Short-Term Memory and Conditional Random Field are utilized for the implementation. Contextual embedding is done with BERT and Glove and the embeddings of these two models are compared in the paper. The results show that BERT embeddings provide better overall results.

Enhancing Clinical Information Retrieval through Context-Aware Queries and Indices [53] emphasizes the importance of context-aware search for clinical information. According to the paper, information retrieval techniques have generally evolved and delivered impressive results, but clinical domain is lacking behind. Especially negation, temporality, certainty and subject are important in clinical context and in defining relevant search results. Custom contextual analyser, trigger removal and contextual relevance scoring are utilized to form relevant ranked results. More specifically, a specific tokenizer, ConText algorithm and modification to the scoring of Elasticsearch are used to achieve the semantic context support and those are implemented to the system as an Elasticsearch plugin.

Semantic Search System using Word Embeddings for query expansion [54] is a general proposition for implementing semantic search with word embeddings. This paper is not a specific case study, but it is still relevant with its implementation descriptions. Each indexed document is also represented as a vector, which is based on the Word2Vec model. Before vectorization, the indexed data is pre-processed by removing stopwords, numbers and strange characters, which do not semantically contribute to the vectors. The query is also pre-processed and vectorized. The vectorized documents and queries are passed to a cosine similarity function, which will provide a synonym expansion for the query. Finally, the expanded query is passed to the search engine, which provides relevant results for the user.

Natural language processing methods for knowledge management — Applying document clustering for fast search and grouping of engineering documents [55] demonstrates a method for searching engineering change request documents. These documents contain unstructured data and there is a lack of solutions, which provide accurate and relevant results with such data. The method includes pre-processing with stop word removal, lemmatization, cleaning and concatenation. After pre-processing the documents are transformed to vectors with Doc2Vec model. Finally, the documents are post-processed by clustering them. The semanticity and relevancy of the system is based on the clustered groups of documents, which will be returned for the user as search results.

As the title of paper *Strings and Things: A Semantic Search Engine for news quotes using Named Entity Recognition* [56] says, it introduces a system, which utilizes data labelling with named entity recognition (NER). The labelling is done with spaCY entity recognizer model and the labelled data is indexed into an Elasticsearch index. Users of the search can use these entity labels by writing them as parts of their queries.

Corpus domain effects on distributional semantic modelling of medical terms [57] utilize clinical, biomedical, and general English domains to gain an understanding on the effectiveness of word2vec model in capturing semantic relatedness between medical terms. The results show that it is slightly beneficial to derive word vectors from in-domain data. Additionally, it is sufficient to utilize open-access biomedical literature when constructing semantic representations of clinical terms. Thus, difficult to access clinical corpora is not required to develop and to evaluate related NLP methods and to utilize them in applications such as semantic search.

4.2.5 Findings

From the selected papers it can be concluded that there is not one definite way of implementing a semantic search functionality. This may be an obvious statement since generally every use case is different and is built around some explicit requirements. However, there are various implementations that all have provided appropriate results in their own field. Some use cases utilize some custom models and algorithms and define equations that optimize the searching process of the case, and some other cases focus on utilizing ready-made tools and models. All in all, it can be said that vector embedding is the generally accepted technique of extracting semantic characteristics from text.

The examined literature includes various methods for vector embedding. Bag of words (BoW) is one of the simplest approaches for vectorization. It captures the frequencies of word occurrences to represent documents. For BoW, a vocabulary has to be defined. The vocabulary also defines the number of dimensions for the vectors. For example, if the vocabulary was [“this”, “there”, “is”, “was”, “a”, “document”] the embedding for text “This is a document” would be [1, 0, 1, 0, 1, 1] – In the text, there is 1 occurrence of “this”, 0 occurrences of “there” and so on. [49]

In early 2010s models such as word2vec and GloVe were developed and introduced to the public. They have the capabilities of vectorizing the semantic meanings of words. The vectors of each word in a document can be combined to represent the meaning of the entire document. [49] As mentioned in Section 2.4, Google’s BERT revolutionized the NLP field. BERT can also be used to generate context-aware embeddings for a semantic search, and it seems like BERT is now the go-to model, or at least one of the most prominent options, for any NLP task. Thus, utilizing BERT embedding, and more specifically FinBERT embedding, is most likely a good approach for case JoutseNet. Additionally, it is probable that indexing metadata such as authors and carefully defining the index structure eases the development work and makes an enjoyable searching experience for the users.

5 Case JoutseNet - Introduction

JoutseNet is a case management system for the personnel of the City of Turku and it has been in use from 1998. Although it is in continuous development, its architecture has not been majorly modernized, which has led to outdated technical architecture and user interface. The users of the system are experiencing significant problems, when trying to utilize its complicated functionalities. All necessary operative actions can still be done with JoutseNet, but efficient use requires extensive knowledge about the case management process and the internal logic of the system. [58]

With JoutseNet, most users want to retrieve information that they need in their everyday work tasks. For example, this information could include decisions or regulations of the city council. JoutseNet is the main tool for the personnel that are responsible for preparing the meetings of boards and other parties. They use the system to compose agendas and transcripts for the meeting sessions.

At the time of writing this thesis, there were three main problems in the JoutseNet system:

- Can be used only with Internet Explorer 11
- The searching functionalities are too complicated and the user needs to extensively understand the system to be able to find relevant information
- The text editor is cumbersome because it is tied to Internet Explorer and uploading and creating documents is problematic for the users

This thesis will focus on implementing better search functionalities for the system. Currently, the search logic of JoutseNet is very outdated, when comparing to modern search engines. The user interface is also too complicated for the users, who have gotten used to easy searching with the search capabilities of Google Search and other industry leaders. The goal is to develop a semantic search functionality, which is easy to use and retrieves relevant search results for the users.

5.1 Case Management Process

Case management is directing the management of cases and related documents of processes in an organization. The management supports the process throughout the whole life cycle of the cases. Its intend is to optimize the preparation, processing, decision making, publishing, archiving, disposal and managing the document information.

The Association of Finnish Municipalities has published a reference framework for case management. [59] It is a general and unified description of processing phases, processes, concepts and metadata, which together form the entirety of case management. When municipalities utilize this framework, they do not have to use resources for describing the basic concepts that are needed as a basis for their case management. The framework also considers the obligations that are set for the case management by legislation. The scope of these obligations is defined by the general laws of information management.

The reference framework is meant to be utilized as a guideline when developing an information system for the case management of a municipality. It is especially needed in the digitisation of administrative procedures. It aims to unify the case management processes and concepts and thereby improve the overall interoperability of implemented solutions. The framework describes how different parts of case management work together. These parts include generalized processes, stakeholders,

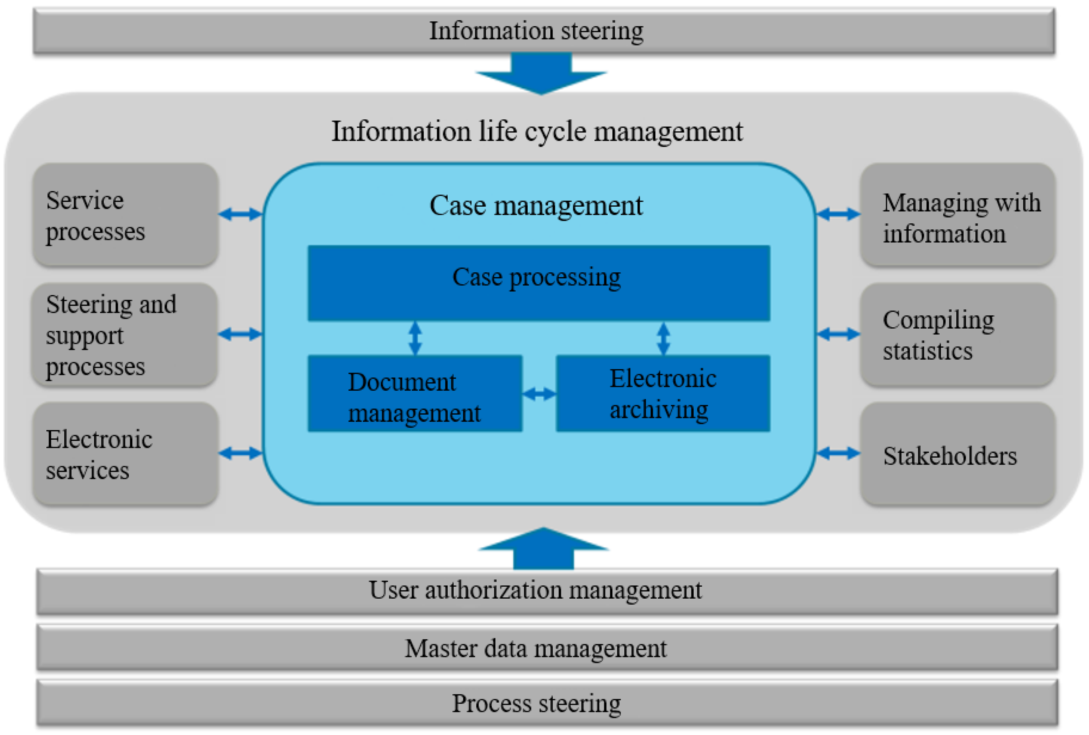


Figure 5.1: Operational environment of case management [59]

roles, information and information system services. This reference framework is used as a guide when defining the desirable implementations and improvements for JoutseNet.

5.2 Operational Environment of Case Management

The Association of Finnish Municipalities has provided a visualization of the operational environment of case management (Figure 5.1) and listed some relevant terms in their framework publication. [59] This chapter will introduce some of those terms, because they are important in understanding the big picture. The original Finnish terms are in parenthesis.

Case (Asia)

Case is an entity, which an official receives or takes for processing. A case can include one or more actions and each action can include documents.

Case Management (Asianhallinta)

Case management is directing the processing of cases and relevant documents throughout their whole life cycle. Case management include case processing, document management and archiving.

Case Processing (Asiankäsittely)

Case processing is a process, which is performed by an official who follows the lawful administrative conduct. The main phases of case processing are becoming pending, preparation, decision making, notification, enforcement and follow-up.

Document (Asiakirja)

Document includes the information that an organization has generated or received while performing an action.

Document Management (Asiakirjahallinta)

Document information management throughout the whole life cycle of the document. The life cycle starts from creating or receiving the document and ends in disposal or permanent archiving of the document.

Information Management (Tiedonhallinta)

Information management is managing the information that an organization acquires, produces, uses or preserves. With proper management and processing the information should be reliable and its utilization should be effective in every action it may be needed.

Information Steering (Tiedonohjaus)

Information steering produces document information system metadata, which is needed for processing and managing the document information.

Information Life Cycle Management (Tiedon elinkaarenhallinta)

With case management, document information is managed throughout its whole life cycle. The phases of information life cycle are creating, saving, using, editing, transferring, copying, sharing, archiving and disposal.

Managing with Information (Tiedolla johtaminen)

Managing with information include the procedures, which are used for refining information and for utilizing information in management of the organization. Information about the organizational actions, operational environment and environmental changes are needed in the management of a municipal organization. Information that is produced and managed in case management should be collected and refined for management purposes.

Archiving (Arkistointi)

Archiving is the permanent or long-term preservation of a document. Archiving follows a predetermined archiving plan.

Service, Steering and Support Processes (Palvelu-, ohjaus- ja tukiprosessit)

In case management, a case can become pending via a municipal service, steering or support process.

Electronic Services (Sähköinen asiointi)

Electronic services are services, which are provided with the help of information and communication technology. For example, with electronic services, a client can follow the status of a case or receive a notification about a decision.

Compiling statistics (Tilastointi)

The information that is produced or managed in case management has to be transferred for statistic compilation. Financial and operational statistics are compiled based on the received information.

Stakeholders (Sidosryhmät)

Stakeholders can start a case management process or produce information or documents for case processing.

User Authorization Management (Käyttövaltuushallinta)

User authorization gives access to information. Managing the authorization is required to ensure the availability, security, originality, integrity, reliability and usability of the information and documents, which are processed in case management.

Master Data Management (Master datan hallinta)

Master data is all the common information, which is utilized in case management by a municipal actor. Employee data and client data are examples of such information.

Process Steering (Prosessiohjaus)

Process steering is steering the actions of case management. It can be performed with an information system.

5.3 User Research

In the context of this thesis, the main purpose of user research is to get an answer to the second research question RQ2: What are the requirements for a JoutseNet semantic search application? Additionally, this section helps to understand the personnel who are using the search functionalities and the environment where those functionalities are used.

For most users, JoutseNet system is a tool, which can be used to fulfil everyday tasks. The system includes various functionalities that can be part of some task, which also requires utilizing some other systems. With user research, it is important to understand the user itself; user is a person who works as a part of their everyday life. The software and systems they are using are only a small part of some larger purpose. Thus, it is important to understand the personas and user stories and use those as guidance for the development of the system.

Section 5.3.1 introduces a persona and a user story, which were created as a part of a preliminary research conducted by ATR Soft Oy. Other personas and user stories have also been created, but they are not relevant when focusing on the search functionality. The personas and user stories have been created by interviewing the users and by observing the usage of the system. The scenarios of Section 5.3.2 have been gathered during the early development of the new search functionality. They expand the understanding on the problems of the current search functionality with a few simple and concrete steps.

5.3.1 User Story: Maritta - Web Chief Editor

Maritta is a fictional persona, which is based on previously gathered research data. [58] She is responsible of writing articles about municipal decisions to the website of the city. For Maritta, JoutseNet is one tool among others and she uses it as a source of information. She represents the majority of the userbase, which use JoutseNet regularly but not in their everyday work. She visits JoutseNet once per month to find something to write about. In the following story, Maritta tries to find all decisions from previous month.

1. Opens JoutseNet

2. Clicks information search button (Tietojen haku) and then decision transcripts button (Päätöspöytäkirjat) from the dropdown menu. New tab opens.

   "I can only use this in one way or at least it is the way that has worked for me. If I do not know the index number (diaarinumero), I start my search with the dates. Index numbers are not usually easy to find."

3. Writes the starting and ending dates to the received (Saapuneet) field.

4. Writes a letter combination to the document type (dokumentin tyyppi) field.

   "How does an ordinary guy in the municipality know the document type? I just happen to know it, but how does he know it?"

5. A new tab opens. All document types are listed there and Maritta clicks the one she is trying to search.

6. Clicks search button (Hae).

7. A new tab opens and it informs that there were zero search results (0 kpl hakutuloksia).

"This looks like there were not any transcripts produced during August." (Which certainly is not the case)

8. Widens the date range.
9. Clicks search button.
10. A new tab opens: zero search results.

    "I think it does not recognize this document type."

11. Tries to modify the search parameters and clicks the search button again. The system gives a notification, which informs that search has already been done with the same parameters.

    "I cannot do it a second time. This got completely stuck now. This is interesting. Are there too many search results?"

12. Changes the document type again and a new tab opens. Gets back to the search page.

    "It would be nice to hear the reasoning for these. Why are there so many options, why do I have to double click here, why is it not steering me?" "Oh now I get it. I have to write the date to the document date field, not to the received field. I do not understand the receiving date. If it is important, why was it not given for the documents I tried to search earlier?"

13. Modifies the document date. (Maritta gets a feeling that she has to fill out all search fields)
14. Clicks the search button.
15. A new tab opens: 2000 search results.

    "This is full of decision transcripts. Now I choose this because it seems interesting."

16. Opens the document by clicking the date of the document in the table of search results. The document opens on an information tab, which is located on a new document search tab. Maritta is a bit frustrated, but she is happy that she finally found something useful.

5.3.2 User Scenarios

User scenarios are specific and brief examples gathered from real users, which have basic understanding of the search functionality and represent majority of the user-base. The scenarios provide background information on some basic information retrieval tasks and on the types of used queries and filters.

Scenario 1 - Searching information about the schedule for filling the market square pit

1. Query string: Market square (Kauppatori)
   (a) Hundreds of irrelevant results. The user decides to filter the search query.
2. Type: Decision (Päätös)
   (a) Lots of results. No results about schedule or plan.
3. "Schedule" (aikataulu) added to query string.
   (a) No search results.

Scenario 2 - Searching decisions related to Aurafest

1. Query string: Aurafest, Date range: 1.1.2021-31.12.2021
   (a) Only one result, which is a document related to selling of tobacco products in Aurafest 2021.

2. The user modifies the date range: 1.1.2018-31.12.2021
    (a) Documents found from previous years, but not from current year (2021).
3. The user is wondering if there is no decision at all for the current year or if the search functionality is faulty.

Scenario 3 - Searching decisions related to the terrace of the cathedral square

1. Query string: Cathedral square (Tuomiokirkkotori)
    (a) No results related to the terrace.
2. Query string modified: Terrace of the cathedral square (Tuomiokirkkotorin terassi)
    (a) No results.
3. Query string modified: Cathedral park (Kirkkopuisto)
    (a) No results related to the terrace.
4. Other attempted query strings: Cathedral park terrace (kirkkopuiston terassi), cathedral terrace (kirkkoterassi), terrace (terassi).
    (a) No results related to the cathedral terrace
5. The user could not find the information they were searching.

5.3.3 Conclusion

This section concludes the preliminary research. The current search functionalities of Joutsenet were examined and user research was conducted to derive a list of requirements (R1-R12) for the semantic search application. Before each requirement, some context is provided in the brief discussion paragraphs.

The user stories and scenarios clarify the fact that the current JoutseNet search functionality is very cumbersome and can even be frustrating to use for the majority of the userbase. For an average user it is common to be stuck with a lot of irrelevant search results or with no results at all. Also, the search happens only after pressing the search-button. Real-time filtering and result listing is missing completely. The users want a fast and clear search functionality that is on par with other modern solutions. It can be quite confidently claimed that an improved functionality can be created by following the modern search application creation principles and adjusting them to the case management system and to the users' needs.

R1. Search results are updated in real-time when a query string is being written or when some filters are used.

R2. The columns of the search result table are sortable.

An optimal search experience provides relevant search results by only taking the query string as an input, but it still is most likely a good idea to include some filters to the functionality. In Chapter 6 the included columns of the search result table will be chosen based on the available data. In addition to the most obvious filters, the users have asked for a decision vacancy filter. Now, the vacancy filter for decision records is inconvenient to use because the user has to know the organization of the vacancy before selecting the vacancy from a list. For example, if dental director is marked to be the decision vacancy for some decision record, the record should be found by giving "dental director" as a vacancy filter.

R3. The columns of the search result table are based on the available data and on the columns of the existing search functionalities of JoutseNet.

R4. There are some filters that are based on the columns of the search result table.

R5. There is a decision vacancy filter in the semantic search application.

Index number is the explicit identifier for the cases in JoutseNet. If the user knows some index number and they want to find the documents under that index number, it should be possible to use the index number as a sole query string or as a filter.

R6. It is possible to search with the index numbers of cases.

Even though the implementation is going to be new and modern, it has to appropriately fit to the functionality and visual templates of the surrounding system. The links in the search result table should match the functionality of the links in the current search functionality to avoid confusion. For example, index number link should direct to the case listing and double click should open the document. The visual side of the semantic search should follow the simplicity and colour palette of the previously implemented user interface for it to be a natural part of the application.

R7. The semantic search application is visually in line with the existing JoutseNet system.

R8. The search results include links, which match the links in the search results of the existing search functionalities of JoutseNet.

From the gathered stories, scenarios, and comments it can be interpreted that it may be quite easy to satisfy the relevancy needs of the users with a new and improved search when the starting level is the very inconvenient search functionality from 1990s. The user research has provided some answers for RQ2, and they are a good starting point for the implementation work: Now it can be said that the users want a search functionality, which is easy to use and provides understandable results with exact keyword matches and can also suggest some additional results based on the semantic similarity. RQ2 will remain as a guiding question in the background during the development and will help to form a more comprehensive and explicit understanding on the topic when there is an application that can be tested by the users.

R9. The semantic search application can retrieve search results with exact or almost exact matches.

R10. The semantic search application can retrieve semantic search results based on the semantic similarities of the query and the indexed data.

Additional Requirements - General Data Protection Regulation (GDPR)

Logging certain actions is overall a good practice for security purposes. On top of that, GDPR requires logging for some occurrences. In the context of JoutseNet semantic search functionality, the logging concerns all decisions and documents that contain sensitive data such as an address or a social security number of some person. In other words, all views of restricted decisions/documents have to be logged. The user can find restricted decisions/documents only if they are authorized to see them with their credentials by belonging to a certain user group. These user groups have to be taken into account when implementing the search functionality.

R11. Finding restricted documents with the search application is possible only if the user belongs to a required user group.

R12. All views of restricted documents are logged.

6 Case JoutseNet - Implementing the Semantic Search

This chapter will go through the implementation of semantic search functionality for JoutseNet case management system and provides a partial answer to the third research question RQ3: How to create a semantic search functionality, which provides relevant search results for the users of JoutseNet-system? The main principles of creating a search engine application introduced in Section 3.2 and the case solutions examined in Section 4.2 are combined and adjusted so that the functionality meets the needs that can be interpreted from the research in Section 5.3.

As mentioned in Section 4.2, Elasticsearch has been selected to be the search engine for case JoutseNet. Elasticsearch is easy to get started with, and it has a well-designed user-interface and documentation. The literature review also shows that Elasticsearch is suitable for various use cases and supports functionalities that enable semantic search. At least in the development phase, Elasticseasrch is installed and run using Docker. Docker is a free application for running software in containers. [60] Elasticsearch docker container is easy to set up by following the instructions provided on the quick start page of Elasticsearch documentation. [61] Kibana, the user interface for managing Elasticsearch data, is also run in a Docker container.

The backend search service is implemented with Python and the search application user interface is implemented with React, which is a free and open-source

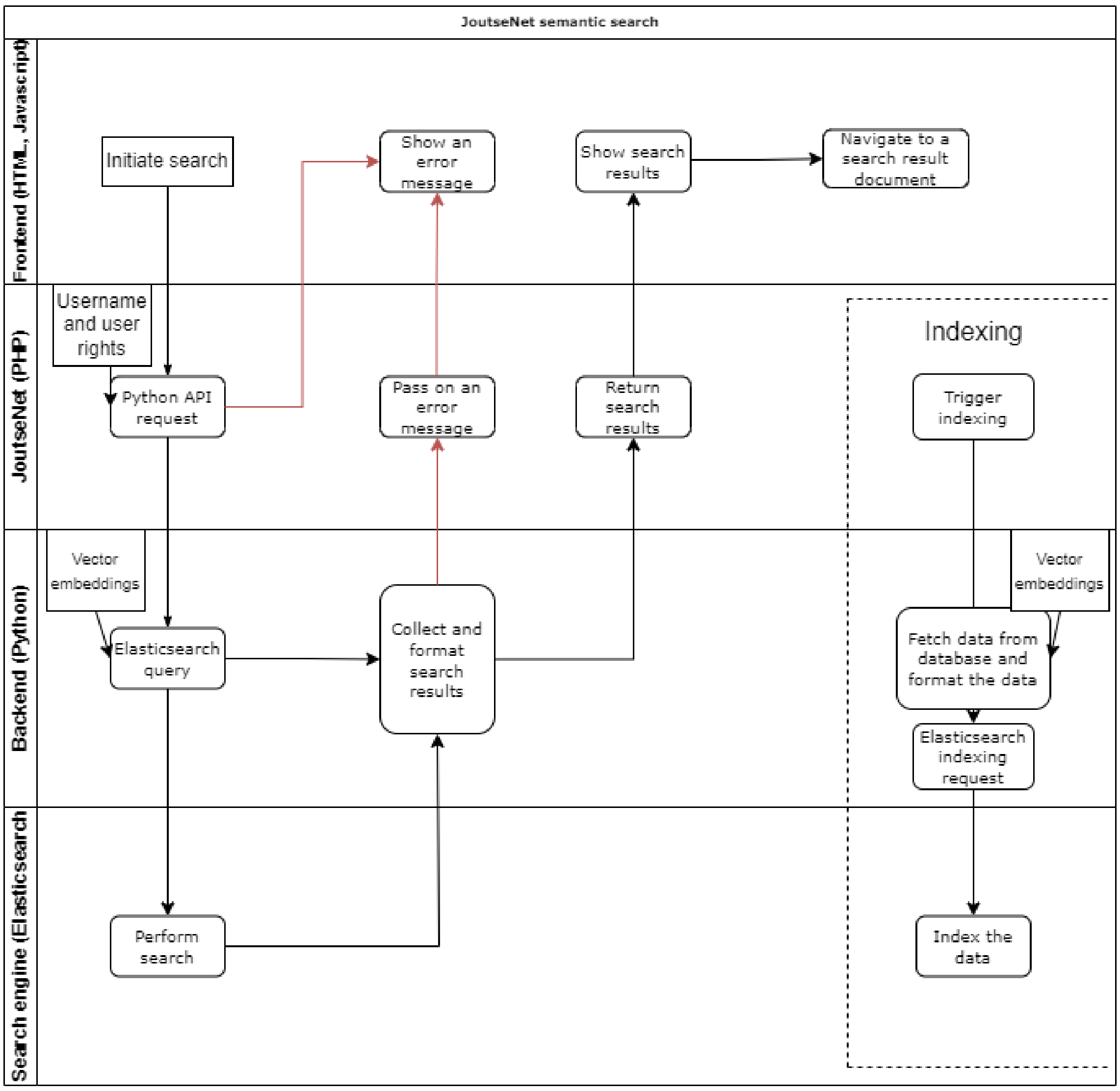


Figure 6.1: JoutseNet semantic search application flowchart

front-end JavaScript library. The current codebase of JoutseNet is mainly implemented with PHP. The semantic search application UI is built and then this React production build is added to the system as an html-template. The semantic search application is developed as a separate element and the goal is to make minimal modifications to the existing codebase. Some additions have to be made to the JoutseNet PHP-application, because usernames and user rights need to be added to the request on the way from the React-frontend to the Python-backend and a ZeroMQ trigger has to be added for enabling cyclic indexing updates.

6.1 Defining the Index Structure

Before indexing anything in the Elasticsearch index, the JoutseNet database has to be examined closely. There are three types of documents that has to be brought from the database to the index: cases (asia), text references (tekstiviite) and files (tiedosto). Cases are the top-level documents, and each index number refers to one case in the system. Cases can include multiple text references and text references can include multiple files. Additionally, each file can be referenced by multiple text references. The database structure is not easy to understand, and it can get complicated with its multiple reference possibilities. However, the most logical way of indexing the data seems to be dividing the index into three parts: cases, text references and files have all their own index. These three document types have their own data fields, so it is convenient to define separate index structures for them. At least in this case, separate indices make the structure easier to manage and it enables straightforward filtering if the user wants to see only one of the three document types in the search result listing.

Next task is to pick the data fields to be indexed. There are some obvious selections such as titles, creation dates, authors and case/text reference/file types. In addition, decision vacancies need to be included for text references and content

```
"mappings": {
  "properties": {
    "dnr": {
      "type": "text",
      "fields": {
        "raw": {
          "type": "keyword"
        }
      }
    }
  }
}
```

Figure 6.2: Example of an Elasticsearch field mapping: Defining a field for the JoutseNet index numbers ("diaarinumero")

of files need to be in the index to make them searchable. As stated in Section 3.2.3, it may also be beneficial to define new field names for the indexed pieces of data rather than just using the field names used in the database. For example, the index field for case title is better as "case_title" than as "title", because the search requests are easier to fine-tune with distinguishable field names. Also, some database fields, such as full versions and abbreviations of document types, can be combined for the index so that users can seamlessly choose which version they want to use in their search.

In Elasticsearch the indexed documents and fields are defined with mapping. There is a schemaless mode/dynamic mapping option for Elasticsearch, but in this case explicit mapping is utilized to gain control on the field structure, names and formats. As mentioned in Section 3.2.2, schemaless mode is a convenient feature to have, but its inaccuracies can outweigh the benefits. Mainly two types of fields, text and keyword, are indexed. Text field type is used for indexing full-text values, which enables automatic analysing and searching for individual words within each full text field. [62] This is the primary field type for all indexed textual JoutseNet data. Additionally, some fields are indexed also as keywords to enable certain sorting and aggregation features. For example, it is useful to aggregate some field values and use the aggregated list as a dropdown selection for some filter. Date data is indexed to date fields to enable appropriate sorting of the values and to have control on the date format. Figure 6.2 shows how the field for JoutseNet index numbers is mapped. Most indexed fields use this same structure. The described selections and decisions in the index mapping have to be made so that it is possible to fulfil requirements R2-R6 and R9 with the backend and frontend implementations. There will most certainly be some modification to the mappings later. The index is constantly shaped and adjusted to surrounding needs during the development work.

6.2 Implementing the Backend

From Section 3.2.3 it can be read that implementing a backend for a search application is probably beneficial and that it is advisable to centralize some operations to the backend. In this case, backend works as a middleman between the frontend and the search index. It receives the search query request, processes it, and forwards it to the Elasticsearch index instance. Elasticsearch conducts the search and returns search results to the backend, which processes the results and forwards them to the frontend so that all necessary data is available for the users and requirement R3 can be fulfilled. Additionally, the data needed for generating the search result links defined in requirement R8 has to be passed to the frontend. The indexing is also initiated and the data processing is done in the backend. Python is the implementation language for the search backend, because it generally is a good choice for data processing and natural language processing. And conveniently, there is an official Elasticsearch client library for Python. [63]

The indexing is done with a python script, which fetches the data from the JoutseNet Oracle-database to the Elasticsearch index. Cyclic indexing updates are implemented with a ZeroMQ queue: There is a trigger for the indexing request in the PHP-code of the JoutseNet application and the receiving end is in the Python-backed of the search application.

Between receiving the data and indexing it to Elasticsearch index, the data needs a little bit of processing. The content of files can be in various formats. There are at least docx, pdf, xml, and html files in the database. The files have to be parsed with appropriate parsers so that it easy to process all contents in the same format. There are helpful tools for this purpose made for Python. For example, pdf-files can be parsed with a tool called pdfplumber. [64]

The backend receives the query string and utilizes it to build a search request. The query string is the most important part of the request but there are many other fields in the request to consider when creating and optimizing it. The queries for Elasticsearch are defined with Query DSL (Domain Specific Language), which is based on JSON. [65] There are multiple query types, and those types have their own subtypes. These different queries should be combined appropriately to achieve the best possible results. For case JoutseNet, function score is used as the top-level query to enable better control of scores with some boosting and additional functions. Native queries of Elasticsearch are utilized to provide search results with exact or almost exact matches and to enable searching with index numbers of cases and other fields that are visible for the users in the user interface. Thus, requirements R6 and R9 are fulfilled with these native queries.

In addition to the main query, some filters need their own queries so that filter selection can be provided for the user and requirements R4 and R5 can be fulfilled:

The user writes something to a filter input and then some possible values for the filter are suggested. When a value is selected for a filter, a new query is formed with the filter and Elasticsearch applies it to provide filtered results. Query examples for the main query and for a filter query can be found in Appendix A.1 - A.2.

As defined by requirements R11 and R12, GDPR logging has to be implemented in the new search functionality. A user may be authorized to find and view some restricted documents if they have the rights for that. In case of finding some restricted documents with the semantic search, timestamp of the search occurrence, username of the user and the ids of the found restricted documents are logged to the JoutseNet database. If the user does not have the needed rights, restricted documents are filtered out when the query is constructed.

6.3 Adding the Semantic Features

As stated in the literature review in Section 4.2, Elasticsearch has some built-in capabilities for supporting vector-based semantic search. Based on previous research reviewed in Sections 2.4, 2.5 and 4.2 it can be assumed that FinBERT embeddings could be a valid approach for a semantic search implementation in a Finnish case management system. Generating the embeddings and searching with them also require some additional work and careful observations on the effects of them. This section describes how to supplement the implementation suggestion with semantic features, which are required by requirement R10.

In February 2022 Elastic published Elasticsearch 8.0 and introduced improvements to vector search capabilities and natural language processing. [66] A new API endpoint, _knn_search, was added and it enabled a native way of utilizing query vectors with approximate nearest neighbour (ANN) search. NLP support was enhanced by the possibility of adding NLP models directly into Elasticsearch. For example, a BERT model could be added directly to Elasticsearch, and the language processing could be performed on the Elasticsearch instance. However, the new endpoint is still in technical preview, which means that its future support is not guaranteed, and separate endpoint makes combining the vector search results and other results inconvenient, since the operations for those cannot be performed under the same query. At the moment, the added NLP support requires a paid platinum subscription, which takes away from the accessibility of these native functionalities. They are good additions, but some other openly available solutions may still be more tempting.

There is also the possibility of using the script score query of Elasticsearch to calculate the measure of cosine similarity between the document vectors and query vectors. [67] Elasticsearch has a built-in "cosineSimilarity"-function for that purpose. This method is suboptimal, because it requires going through the entire index and calculating the angular distances for all documents.

Alex Klibisz, an American software engineer, has developed an Elasticsearch plugin called Elastiknn, which provides "efficient exact and approximate vector search to Elasticsearch". [68] It fills the gap formed by the limited native support for vectors and enables combining traditional queries with vector search queries. Elastiknn has quite comprehensive and clear documentation and it seems to serve well the purposes of JoutseNet semantic search, so it is selected as the implementation technique for the case. Elastiknn version 7.17.4 is used for the development since that was the latest version when the development was started. The used version of Elasticsearch must match the plugin's version so the version of Elasticsearch is also 7.17.4 for this project.

There are multiple exact nearest neighbour query types and approximate query types to choose from in Elastiknn: L1 (taxicab geometry), L2 (Euclidean distance), cosine similarity, jaccard index and hamming distance. It is not necessary to include the explanations for all these functions and mathematical theories in this thesis.

```
model = SentenceTransformer('TurkuNLP/sbert-cased-finnis-paraphrase')
query = ['Turun satama']
query_vector = model.encode(query)
```

Figure 6.3: Example of embedding a query string, which is intended to find information about or related to Port of Turku

Based on previous literature and experiences, cosine similarity is determined to be the preferred function for the implementation of the case. Simply put, cosine similarity is based on the angle between two vectors. If the angle is small, the vectors are very similar, and the represented documents are similar semantically. [54] In this case, the compared vectors are the query string vectors and the indexed vectors.

Before the vector search is possible, the vector embeddings have to be generated somehow and those have to be added to the indexing process. Fortunately, the embedding implementation is quite straightforward with today's ready-made tools and libraries. FinBERT seems to be a good model option for the case since it is optimized for Finnish language. TurkuNLP has also trained Finnish Sentence BERT from FinBERT and it is available on Hugging Face [69], which is a data science platform for building, training and deploying ML models. The sentence model can be utilized with a Python framework called SentenceTransformers. [70] Figure 6.3 shows how simple SentenceTransformers is to use : The chosen model is downloaded with a single line of code and then some textual data can be embedded with another line of code.

In case JoutseNet, the titles of cases and text references, the contents of files and the query strings are embedded. Embedding the titles and query strings is a simple task because they are relatively short pieces of text. However, embedding the contents of files brings a bit of challenge and it requires some further consideration.

The maximum sequence length for FinBERT, and most BERT models, is 512 tokens. Thus, the model can only embed strings that are not more than 512 tokens long. In Finnish and with FinBERT sentence model 512 tokens is roughly 250 words. The files in JoutseNet can be very long and easily exceed that limit. Therefore, a separate solution for embedding the long documents has to be implemented.

Techniques and tools for embedding words and sentences are widely available and they can provide quite impressive results. Document embedding has also been researched for a few years already, but it is still a task that requires a bit more experimental approach. Documents have different structures and content in different systems, which amplifies the fact that there is no solution that can be presented as a universal answer for the problem. Some data scientists have written prominent and helpful blog posts in which they have presented the current possibilities for document embedding. [71] [72] From these previous experiences it can be read that there are at least three types of implementations that have provided acceptable results: slicing off the document parts that go beyond the 512 tokens, summarizing the document and taking the average of vectors, which have been encoded from some smaller chunks extracted from the documents. Slicing is the easiest way, but it possibly provides inaccurate results because a large part of the documents may be completely ignored. Therefore, either summarization or averaging is proposed to be the implementation technique for case JoutseNet. Implementation suggestions for both techniques are provided, and some testing is utilized to determine which one is the one to proceed with. Example code snippets for the implementation suggestions can be found in Appendix B.1-B.2. Papers reviewed in Section 4.2 point out that it is a good practice to preprocess the documents by excluding the semantically irrelevant parts. [54] [55] Some basic cleaning is done for the file contents before embedding them: digits and some other redundant parts, such as empty strings, are removed because they do not provide practical value for the vector embeddings.

```
"vector_field": {
  "type": "elastiknn_dense_float_vector",
  "elastiknn": {
    "dims": 768,
    "model": "lsh",
    "similarity": "cosine",
    "L": 99,
    "k": 1
  }
}
```

Figure 6.4: Example of Elastiknn mapping

The index mapping and query need to be modified to make the semantic features functional. There are two vector types in Elastiknn: elastiknn_sparse_bool_vector and elastiknn_dense_float_vector. The sparse bool vector type is intended for vectors where the indices are either true or false and majority of them are false. Only true indices are stored to save space. This datatype is suitable especially for bag-of-words approach. The dense float vector type is optimized for vectors with floating point numbers. Each number is a value of one vector dimension and in optimal case there are no more than 1000 dimensions. The embeddings produced with FinBERT have 768 dimensions. Thus, the dense float vectors type is suitable for storing the embeddings of titles and contents of case JoutseNet. To select the cosine lsh mapping as indexing type, the model is set to be "lsh" and the similarity is "cosine". From mapping snippet in Figure 6.4 it can be seen that there are also values L and k to be determined. L is the number of hash tables and k is the number of hash functions combined to form a single hash value. Increasing the value of L increases recall and increasing the value of k increases precision. Higher recall means

```
{
    "elastiknn_nearest_neighbors": {
        "field": "vector_field",
        "vec": {
        "values": query_vector
        },
        "model": "lsh",
        "similarity": "cosine",
        "candidates": 100000
    }
}
```

Figure 6.5: Example of Elastiknn query

that higher percent of all relevant documents are found. Higher precision means that higher percent of the found documents are relevant. [73] These two are competing objectives so it is beneficial to set a high value for L and low value for k to get as many relevant search results as possible. Finding the optimal mapping require testing with various combinations of values.

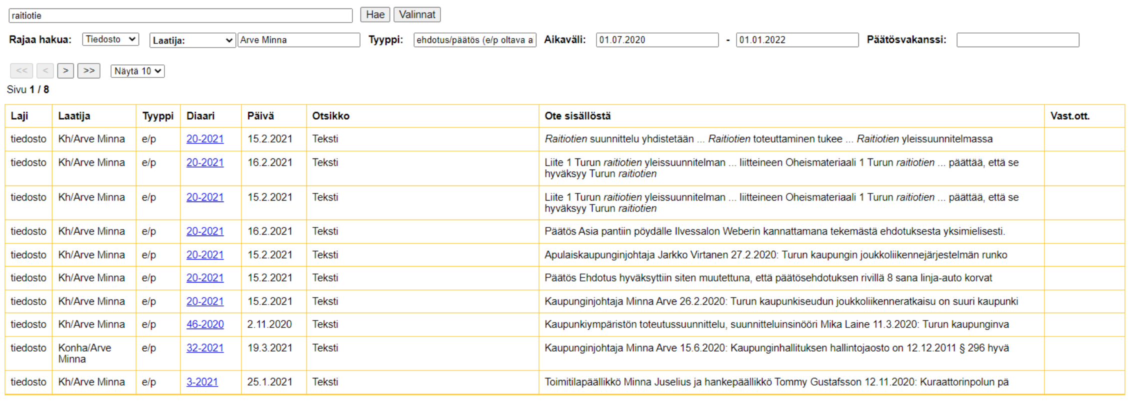
Using and constructing the Elastiknn query is quite simple and convenient since it can be combined with the native Elasticsearch queries. It is just one query type more on the side of the multi match queries or other native queries that have been chosen to be used. The Elastiknn query require defining the field of the vectors, the value of the query vector, the model type, the similarity type, and the number of candidates. Elastiknn first retrieves approximate nearest neighbours to the query vector and then computes exact similarities to the number of documents that is defined by the candidates value. Increasing the number of candidates increases recall but it also increases latency. The goal is to determine as high number of candidates as possible and to keep the latency acceptable.

6.4 Implementing the Frontend

Frontend defines the user interface for the application. Based on requirements R1-R5, R7 and R8 defined in Section 5.3, the goal is to create a modern UI with responsive and reactive capabilities while keeping it visually familiar for the users. It is natural for the users to define requirements by expressing what they want to have in the UI. The importance of UI in guiding the other parts of the implementation is discussed in Section 3.2.3. In Case JoutseNet, the users' needs for specific filters and other UI partitions generate requirements for the backend, which includes the functionality for defining and retrieving the information that is shown in the UI.

As mentioned in the beginning of this chapter, the implementation library is React. It is one of the most popular libraries for building user interfaces and it provides tools for quite effortless building of a separate component and integrating it to an existing system. A react component is most probably easy to maintain in the future because many developers know it, it has a clear and well-maintained documentation and the community actively develop and maintain component packages for various UI purposes.

The frontend is developed with four JavaScript-files: App.js collects all the components together and defines the search functionality and the columns for the search result table, Styles.js defines the styles for the whole application, Table.js defines the structure of the search result table and index.js renders the App-component to the root element. Additionally, gulpfile.js determines how the application build is transformed into a single index.html file, which can be integrated to the existing JoutseNet source code. The input field states and indicators are implemented with React state hooks [74], the result table is implemented with React Table library [75] and the styles are implemented with styled-components [76] library. To make



raitiotie

Hae Valinnat

Rajaa hakua: Tiedosto **Laatija:** Arve Minna **Tyyppi:** ehdotus/päätös (e/p oltava a **Aikaväli:** 01.07.2020 - 01.01.2022 **Päätösvakanssi:**

<< < > >> Näytä 10

Sivu **1 / 8**

Laji	**Laatija**	**Tyyppi**	**Diaari**	**Päivä**	**Otsikko**	**Ote sisällöstä**	**Vast.ott.**
tiedosto	Kh/Arve Minna	e/p	20-2021	15.2.2021	Teksti	*Raitiotien* suunnittelu yhdistetään ... *Raitiotien* toteuttaminen tukee ... *Raitiotien* yleissuunnitelmassa	
tiedosto	Kh/Arve Minna	e/p	20-2021	16.2.2021	Teksti	Liite 1 Turun *raitiotien* yleissuunnitelman ... liitteineen Oheismateriaali 1 Turun *raitiotien* ... päättää, että se hyväksyy Turun *raitiotien*	
tiedosto	Kh/Arve Minna	e/p	20-2021	15.2.2021	Teksti	Liite 1 Turun *raitiotien* yleissuunnitelman ... liitteineen Oheismateriaali 1 Turun *raitiotien* ... päättää, että se hyväksyy Turun *raitiotien*	
tiedosto	Kh/Arve Minna	e/p	20-2021	16.2.2021	Teksti	Päätös Asia pantiin pöydälle Ilvessalon Weberin kannattamana tekemästä ehdotuksesta yksimielisesti.	
tiedosto	Kh/Arve Minna	e/p	20-2021	15.2.2021	Teksti	Apulaiskaupunginjohtaja Jarkko Virtanen 27.2.2020: Turun kaupungin joukkoliikennejärjestelmän runko	
tiedosto	Kh/Arve Minna	e/p	20-2021	15.2.2021	Teksti	Päätös Ehdotus hyväksyttiin siten muutettuna, että päätösehdotuksen rivillä 8 sana linja-auto korvat	
tiedosto	Kh/Arve Minna	e/p	20-2021	15.2.2021	Teksti	Kaupunginjohtaja Minna Arve 26.2.2020: Turun kaupunkiseudun joukkoliikenneratkaisu on suuri kaupunki	
tiedosto	Kh/Arve Minna	e/p	46-2020	2.11.2020	Teksti	Kaupunkiympäristön toteutussuunnittelu, suunnitteluinsinööri Mika Laine 11.3.2020: Turun kaupunginva	
tiedosto	Konha/Arve Minna	e/p	32-2021	19.3.2021	Teksti	Kaupunginjohtaja Minna Arve 15.6.2020: Kaupunginhallituksen hallintojaosto on 12.12.2011 § 296 hyvä	
tiedosto	Kh/Arve Minna	e/p	3-2021	25.1.2021	Teksti	Toimitilapäällikkö Minna Juselius ja hankepäällikkö Tommy Gustafsson 12.11.2020: Kuraattorinpolun pä	

Figure 6.6: The user interface of JoutseNet semantic search

the application work with Internet Explorer, some polyfills have to be added to the index.js and some inputs and data lists have to be implemented with jQuery.

Figure 6.6 shows the first proposed version of the search UI. The simple characteristics and the yellow colour are in line with the existing JoutseNet UI, which satisfies requirement R7. To fulfill requirement R1, a search request is sent to the backend every time a letter is written to the search box or the state of some filter is changed, but there is also a search-button (“Hae”), which can be used to send the request. The options-button (“Valinnat”) makes the filters visible when pressed. The filters are not visible when first arriving to the search page. At the moment there are five filters that are implemented to fulfill requirements R4 and R5: Limit the search -filter (“Rajaa hakua”) gives the option to only show cases, text references or files. With the second one, the user can select from a dropdown menu if they want to filter with the author (“Laatija”) or with the recipient (“Vastaanottaja”). The type-filter (“Tyyppi”) is for the different types of documents. Lastly, there are date range (“Aikaväli”) and decision vacancy (“Päätösvakanssi”) filters. Author, recipient, type and decision vacancy filters all give suggestion below them on a dropdown menu and the user selects the desired option from that menu. For the type filter, abbreviations and full versions are both accepted.

The result table is paginated, and the page can be switched from the buttons above the table. The user can also select if they want to see 10, 20, 30, 40 or 50 results on one page. The ability to sort search results is required by requirement R2: All column headers, except the content column (“Ote sisällöstä”), can be used for sorting by clicking the header. A result row is highlighted with a yellow colour when mouse is hovered over it. To save some space, the types are shown only as abbreviations in the table, but the full version is shown as a tooltip, when mouse is hovered over the type of some result row. Double clicking a result directs to the corresponding document. Clicking the index number (“Diaari”) of a result always directs to the case listing, which includes all documents that are under the case, which the index number is assigned to. The document and case listing views are existing views in the JoutseNet system. The semantic search application only generates links to those views and attaches them to the results. These links match the links provided by the existing search functionalities of JoutseNet, as defined by requirement R8. To give an indication about the state of the system for the user, a “Loading the search results” -message (“Ladataan hakutuloksia”) is shown below the search options. If there are no search results, a “No search results” -message (“Ei hakutuloksia”) is shown and in case of an error, there is a “Search server does not answer” -message (“Hakupalvelin ei vastaa”).

7 Case JoutseNet - Testing and Optimizing the Semantic Search

This chapter provides some ideas and guidelines for testing the implemented semantic search functionality with production data and continues answering to RQ3: How to create a semantic search functionality, which provides relevant search results for the users of JoutseNet-system? When the development work and the writing process of this thesis started, the initial plan was to test the functionality with the production data and to document the results to this chapter. Unfortunately, getting the production database turned out to be a prolonged process, because the database includes sensitive data. Processing the data to be suitable for research purposes would have required allocating extensive tasks to the development personnel of the system. This led to the decision of finishing the thesis and the current development phase without the production data. Test database has been available from the beginning of the development, but it is valid only for testing the database connection and the indexing process. The contents of the cases (asia), text references (tekstiviite) and files (tiedosto) are not realistic at all in the test database and the size of the test database is much smaller when compared to the size of the production database. The basic functionality of the search application can be verified with the test database, but further analysis is unnecessary because the indexed data does not provide realistic results and it is impossible to evaluate the relevancy of the results.

7.1 Optimizing the Indexing Process and the Query Request

Before testing the search itself, it has to be ensured that the indexing process defined with the help of the test data is functional also with the production data. The amount of data in the production database is quite immense. Thus, the indexing process is better to test first with a small sample of the data. The overall structure of the indexing process should not cause any problems since the test and production databases supposedly have the same structure. However, the formats and contents of files most probably vary quite a lot, and some unexpected errors could be encountered during the parsing and embedding process before they are ready to be indexed. As mentioned in Section 6.3, some cleaning is done for the file contents before they are embedded. It should be checked if the suggested cleaning of digits and other redundant characters is enough or if some modifications are needed to include only the relevant information for the vector embeddings. Logging and observing the parsed file contents is a good way to gain understanding on the contents as raw text and on the embedding process.

The initial indexing of all production data may be a lengthy process because of the large number of documents. Additionally, the embedding of the file contents probably lengthens the overall indexing process significantly if the processing power is limited. The embedding is done by the graphics processing unit (GPU) of the system, so it is beneficial to have a development environment with a GPU, which is sufficiently capable in needed NLP tasks. The processing power is essential especially during the iterative and experimental development phase, because re-indexing of large numbers of documents is needed after modifications are made to the index mappings or to the overall process. The processing power is less relevant in the production environment, because the large initial indexing has to be done only once and after that new documents are fetched from database to the index with cyclic updates.

The most relevant parts for adjustments in the indexing process are the embedding functions and the index mappings. As mentioned in Section 6.3, different values for parameters L and k for the vector index mappings has to be tested so that optimal values can be defined. The initial suggested values (L: 99 and k: 1) are picked from the example in the Elastiknn documentation [68] and they are a good starting point for the testing as the goal is to have a low value for k and a high value for L. The optimal values for these parameters are different for every use case.

Two embedding technique suggestions are given in this thesis: summarization and averaging. They both have their own possibilities for adjustments. The length of the summary in the summarization alternative can be easily modified by adjusting the number of sentences used for forming the summary. For the averaging alternative, the length of the chunks can be modified when trying to find the optimal embedding function. The embedding functions could be sufficient in their suggested forms, but they should not be accepted without experimenting with different values, when optimizing the search functionality.

The query request is obviously a vital part of the search application, and it probably is quite challenging task to optimize it for the use case. The given query request suggestion uses function score as the top-level query type and uses multi match queries to enable some additional features for the query. The adjustable values, such as boost, slop and fuzziness, of the different multi match queries should be adjusted when testing and optimizing the search functionality. Some additional query types could be more optimal, and some suggested queries may turn out to be completely redundant. For example, the fuzziness parameter of the best fields query could be unnecessary and cause irrelevant search results if the query string is too vague after the fuzziness parameter is applied. Furthermore, if the further analysis of the search application needs to distinguish the effect of the semantic features from the other search features, it may be beneficial to retrieve the most relevant search results with exact matches and complementary results with semantic features. The suggested main query combines all subqueries under one larger query and the significance of individual subqueries can be adjusted with their weight values.

7.2 User Testing

Like generally in software development, potential users and other key personnel should be constantly involved in the development and testing process of a search functionality. A lot of time and effort could be wasted if the implementation and optimization is based on assumptions without including domain and content expertise. During the early development of the JoutseNet semantic search, the user interface and the responsiveness of the functionality have been demonstrated to a few users and the initial reactions have been very positive. It can be already said that the functionality is seemingly on the right path, but actual testing of the search functionality and feedback on the search results have not been possible with the provided test database. Even if there was a production database or some accurate representation of it, it would not be enough to perform testing without the input of representatives from the userbase.

With the characters in Figure 7.1, Turnbull and Berryman demonstrate the roles of the involved personnel. [27] In this scenario, the users are not content with the search results, the manager notices the frustration and the domain expert can explain why the search results are irrelevant. The content curator is aware of the overall situation, which includes the content that is searched, the customers and the business needs. Together with the content curator, the search engineer implements adjustments to the search functionality so that it answers better to the business needs and the users find it valuable. Content curator can be a permanent title for

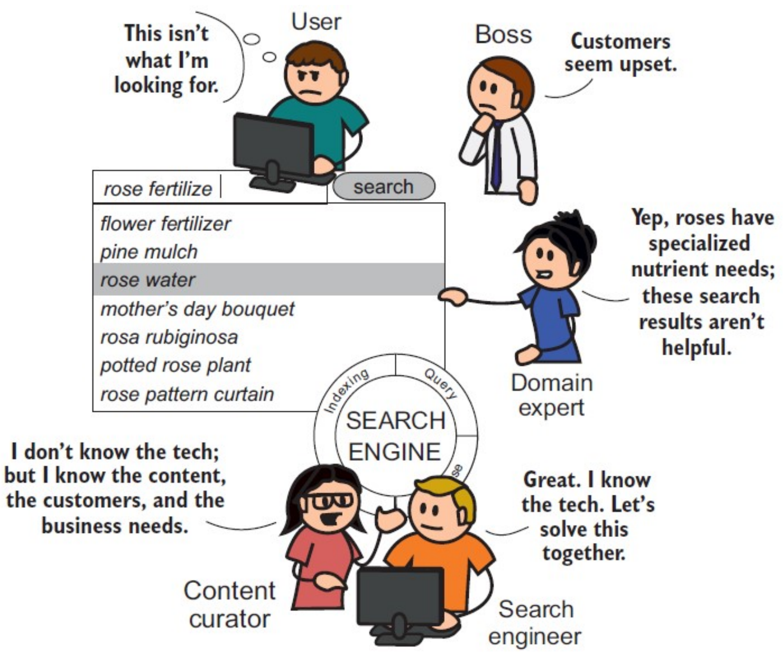

This isn't what I'm looking for.
User
Boss
Customers seem upset.
rose fertilize
search
flower fertilizer
pine mulch
rose water
mother's day bouquet
rosa rubiginosa
potted rose plant
rose pattern curtain
Yep, roses have specialized nutrient needs; these search results aren't helpful.
Domain expert
Indexing
Query
SEARCH ENGINE
I don't know the tech; but I know the content, the customers, and the business needs.
Great. I know the tech. Let's solve this together.
Content curator
Search engineer

Figure 7.1: People involved in relevance tuning [27]

someone, but in many cases, it is someone who is appointed to temporarily to have the appropriate responsibilities. Content curator should be someone who has the sufficient expertise and seniority to own the correctness of the search. According to Turnbull and Berryman, the content curator can be defined as the product owner of the search application.

Having a content curator, and a domain expert, makes understanding the feedback and users more convenient, but individual users have the most important role: They are the ones who actually do the testing and generate the raw feedback for the development. Iterative development process enables delivering a real solution quickly for evaluation. Figure 7.2 presents the simple but effective iteration for adjusting search relevance. First, the search application is deployed to some environment, where users can test it and generate feedback. After some testing, the development team analyses the feedback and the relevance of the search results. From that analysis the reasons behind the dissatisfaction of the users can be derived to some extent and adjustments can be made. Defining the search relevance

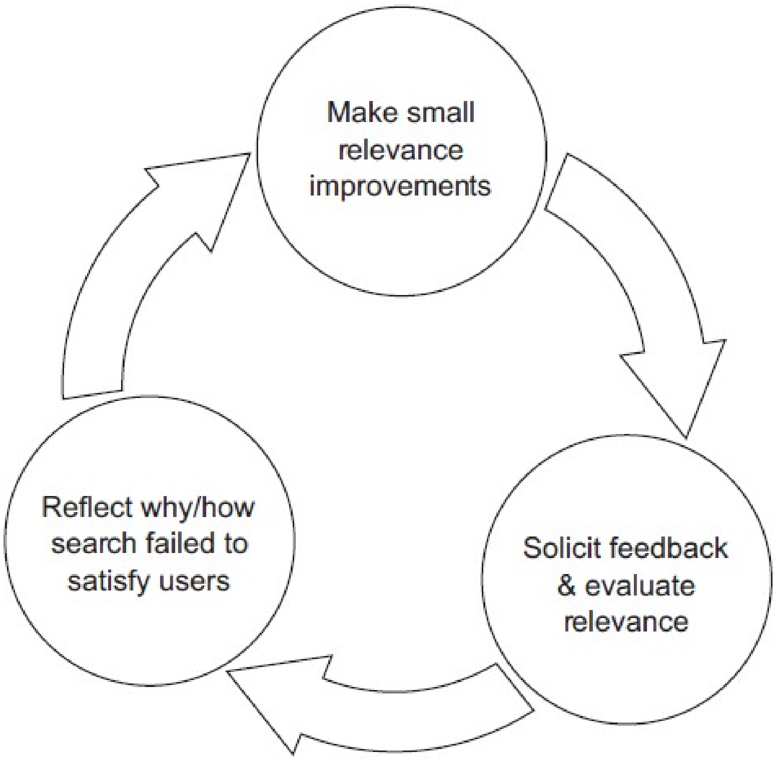

Make small relevance improvements
Solicit feedback & evaluate relevance
Reflect why/how search failed to satisfy users

Figure 7.2: Relevance feedback loop [27]

for some specific business need is a difficult task and the first version of a search functionality may not deliver results that are expected by the users. Sometimes a fortuitous guess can provide good results, but some other guesses could lead to unfortunate situations. Search engineers should not be afraid of failing fast and adjusting with quick iterations. With small improvements it is easier to steer the functionality to right direction and to avoid worse failures. The goal is to have a user-focused organizational culture, which is committed to deliver accurate feedback to the engineers and curators improving the search results and the application as a whole. [27]

7.3 Next Steps for Case JoutseNet

This section describes the suggested configuration and testing steps before deploying the created search functionality to the production environment. As can be read from Section 7.1, the implementation includes multiple parts that can be adjusted when optimizing the indexing process and the query request.

Pilot testing is suggested to be the method for delivering the first version of the semantic search implementation to the users: The semantic search application is deployed to a test environment, where users can test it. The data used in the test environment should be as realistic as possible, preferably the production data, so that the relevancy of the search results and the generated feedback are realistic, and the behaviour of the application is identical or very close to how it would be in the production environment. The iterative testing process introduced in Section 7.2 is suggested to be utilized when performing the pilot testing.

Before deploying the application to the test environment, the compatibility with the current JoutseNet version has to be ensured, because the system is in continuous development. The semantic search implementation introduced in this thesis has been developed in isolation from other development activities of JoutseNet. Thus, the semantic search UI and other small additions made to the JoutseNet source code may need to be updated.

To compare the two suggested embedding alternatives (summarizing and averaging), it would be convenient to have the possibility to index the data with both embedding methods to separate indices. Then search results provided by both alternatives could be examined and compared with each other in the test environment. It is a good idea to experiment with the adjustable values of the indexing and embedding process, most importantly the ones mentioned in Section 7.1, with a few testing iterations. If one of the alternatives starts to show significantly better results, it can be chosen for further development and the other can be left out. Especially for evaluating the semantic search results found with the vector search, collecting some realistic search queries from the users, and using the same queries iteratively with the users, could help keeping track of the result changes after making some adjustments to the embedding/index mapping values. During the development and testing process, it has to be remembered that adjusting these values only affects results retrieved with the Elastiknn-query, not the results retrieved with the other simultaneous queries, such as multi-match-queries.

Adjusting the query is probably the part that takes the most effort and iterations in the optimization process. It is beneficial to let the users test the search and generate feedback freely so that the used search strings, filters etc. are as realistic as possible, but there should also be some thoroughly documented user scenarios that can be analysed together with the users. These user scenarios can be utilized to generate comparable results between the iterations. Even though the semantic features of the application largely define the overall functionality, they are only a part of the implementation. Semantic search results are based only on the titles of the cases and text references and on the contents of the files. There are many other simpler fields, such as index numbers and types of documents, that are searched only with the native Elasticsearch queries. When optimizing the query, the significance and weights between the native queries and the vector search query have to be solved: Is it enough to have exact matches with the native queries or are some advanced features of the queries needed to complement the retrieved results? Should the results retrieved with the native queries always be first in the result list and after them the complementary semantic results or is some other more heterogeneous approach better? As mentioned in Section 7.1, distinguishing the results retrieved with the native queries and with the Elastiknn-queries could be beneficial for analysis purposes, but the desired structure of search results in practice could be something completely different and it can be validated only with appropriate user testing.

Search result relevance is the most central part of the suggested search application, but the user interface is obviously an important part of the user experience, and its optimization and development should not be forgotten in the testing process. If there is something wrong with the user interface, the users will probably have comments on the deficiencies. The suggested user interface is made to visually fit the surrounding JoutseNet system, and the functionality of links embedded into the search results are mimicked from the present search functionalities of the system. Filters are based on previous functionalities and on the initial wishes made by the users during the early development of the semantic search application. The users should determine if the links, filters and the user interface as a whole are working as expected or if something needs to be modified, removed or added.

8 Conclusion

8.1 Answers to Research Questions

RQ1: What kind of methods and techniques are used in previous use cases to enable semantic search functionalities?

As stated in Chapter 4, the purpose of RQ1 is to collect ideas and techniques for the implementation of the JoutseNet semantic search. A Systematic Literature Review (SLR) was utilized to retrieve and review relevant studies regarding previous use cases on semantic search. The most important learnings of the literature review are the methods and techniques for the implementations, not the characteristics of the use cases. However, including the use cases in the research question and in the review process appropriately limited the search results to implementations with a specific scope, which are relevant when designing the JoutseNet semantic search. Some of the reviewed papers do not include a specific or a strictly limited use case, but they are still relevant with their implementation techniques and examples.

The literature review emphasizes the fact that it may not be an easy task to select the techniques for a semantic search and finding the correct tools may require extensive experimenting. Generally, vector embedding is a widely utilized technique for bringing semantic features to search applications. GLOVE word vectors, Word2Vec, Doc2Vec, BERT and even some custom embedding methods are used to vectorize pieces of text. Depending on the use case, the vectors are utilized to calculate angular distances between queries and documents, to cluster documents or to provide synonym expansions for queries. However, vector embedding is not the only way to implement semantic search: bag of words, weighing the documents, custom context algorithms and named entity recognition are also sufficient to achieve some level of semantic functionalities. Additionally, some more fundamental NLP processes, such as stop word removal, lemmatization and tokenization are beneficial when preparing the documents.

RQ2: What are the requirements for a JoutseNet semantic search application?

Conducting user research and examining the current search functionalities were utilized in Section 5.3 to derive some requirements and guidelines for the semantic search application. The high-level requirement is to have a modern and responsive application, which is easy to use and which retrieves relevant search results for the user of JoutseNet. The relevant results can be retrieved with exact or almost exact matches, but they should be supplemented with semantically similar results. The goal is to have a “Google-like” intuitive search, but the users want to have a few filters, such as document type and date filters, so that it easy to be in control of the searching process, especially if the initial string query provides a large number of results. Index number of the cases in JoutseNet-system is an important data field and it must be possible to search for documents with an index number.

Regarding the search experience and performance, the requirements drive the application to be modern and close to the current industry standards, but the user interface still must be kept more traditional so that the overall visuals and the functionality of the search result links do not differ from the surrounding system and the patterns of the users. These initial requirements help guiding the early development of the application, but they are not enough to define a detailed implementation.

It seems like it is quite difficult for the users to express their needs before they are given an example solution that they can test. Thus, the requirements will be enhanced during further development. Especially requirements that steer the application to retrieve relevant search results are defined when iteratively testing the initial implementation with users.

RQ3: How to create a semantic search functionality, which provides relevant search results for the users of JoutseNet-system?

The technical implementation suggestion documented in Chapter 6 partially answers to RQ3. The implementation includes a backend implemented with Python and a frontend implemented with React. Some modifications are made to the PHP source code of JoutseNet for enabling the user interface of the semantic search as part of the surrounding system, for passing user rights and for triggering the indexing of new documents added to the database. Elasticsearch is selected to be the search engine in the implementation and Elastiknn-plugin enables semantic features in the Elasticsearch instance. FinBERT vector embeddings are utilized to enable semantic search for the titles of cases and text references and for the contents of files. Two implementation suggestions for embedding the file contents are given: summarization and averaging. Example code snippets for these suggestions can be found in Appendix B.1-B.2. The vector embeddings are indexed to the Elasticsearch index as Elastiknn dense float vectors and all other necessary data fields are indexed with the native mapping schemas of Elasticsearch. Search results are retrieved with a query, where Elastiknn nearest neighbor query is combined with the native queries of Elasticsearch. Examples of queries used in the implementation can be found in Appendix A.1 - A.2.

Chapter 7 continues answering to RQ3 by providing suggestions on how to test and optimize the application and by describing the concrete next steps in the development process. Various parameters, such as the ones defined in the vector index mappings, can be adjusted when experimenting with the application. An iterative pilot testing process is suggested to be the method for adjusting the queries, the user interface and all other optimizable parts of the application. The initial plan for the thesis was to validate the functionality of the application with the production data. Without the production data it had to be decided that a more theoretic discussion of the further development process is enough, even though RQ3 is not fully answered with validated results.

8.2 Discussion and Future Work

The outcome of this thesis is an implementation suggestion for a semantic search application with two possible alternatives for vector embedding, which enable semantic search results. Generating sufficient vector embeddings for short pieces of text is rather easy with today's tools and frameworks. However, doing the same with longer pieces of text can still turn out to be relatively challenging, especially when the used language is a small language such as Finnish. Summarization and averaging for long text documents are suggested to be solutions for getting around the limitations of BERT. Different kind of an approach to the problem could be creating a completely new long model, which is optimized for Finnish language and which accepts longer token sequences to be encoded at a time. For example, building and training a Longformer [77] model for Finnish could provide solution possibilities for many Finnish organizations.

Creating an optimized search application is not an easy task. Hopefully, the suggested implementations for JoutseNet semantic search are steering the development to right direction and provide value for the users of the system. Methods for the implementation and optimization have been collected from various sources and they are documented in the theory parts of this thesis. Especially book *Relevant* *search: With applications for Solr and Elasticsearch* by Turnbull and Berryman [27] has provided guidelines for developing a search application and for involving users in the testing process. It is recommended to take a look at the book's relevance framework when trying to solve the relevancy problem of a search application.

In any case, there is probably still a lot to be done with the JoutseNet semantic search before it can be considered to be a viable solution: An unspecified number of testing iterations are needed to optimize the application and to sufficiently answer to the users' needs. Further research on the new search application could be conducted by examining the provided value for the users. First indications of the value could be evaluated during the initial testing of the application and the value measurement process could be continued in the production environment by monitoring the behaviour and feedback of the users. As an existing system, JoutseNet enables comparative research by having the possibility of comparing new implementation suggestions to the ones previously implemented.

References

[1] E. Kumar, *Natural Language Processing*, en. I. K. International Pvt Ltd, Dec. 2013, ISBN: 978-93-80578-77-4.

[2] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: An introduction", *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, Sep. 2011, ISSN: 1067-5027. DOI: `10.1136/amiajnl-2011-000464`. [Online]. Available: `https://doi.org/10.1136/amiajnl-2011-000464` (visited on 05/27/2022).

[3] D. Mollá and J. L. Vicedo, "Question Answering in Restricted Domains: An Overview", *Computational Linguistics*, vol. 33, no. 1, pp. 41–61, Mar. 2007, ISSN: 0891-2017. DOI: `10.1162/coli.2007.33.1.41`. [Online]. Available: `https://doi.org/10.1162/coli.2007.33.1.41` (visited on 05/30/2022).

[4] T. P. Nagarhalli, V. Vaze, and N. K. Rana, "Impact of Machine Learning in Natural Language Processing: A Review", in *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, Feb. 2021, pp. 1529–1534. DOI: `10.1109/ICICV50876.2021.9388380`.

[5] M. Fares, S. Oepen, and Y. Zhang, "Machine Learning for High-Quality Tokenization Replicating Variable Tokenization Schemes", en, in *Computational Linguistics and Intelligent Text Processing*, A. Gelbukh, Ed., ser. Lecture

Notes in Computer Science, Berlin, Heidelberg: Springer, 2013, pp. 231–244, ISBN: 978-3-642-37247-6. DOI: 10.1007/978-3-642-37247-6_19.

[6] R. Navigli, "Word sense disambiguation: A survey", en, *ACM Computing Surveys*, vol. 41, no. 2, pp. 1–69, Feb. 2009, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/1459352.1459355. [Online]. Available: https://dl.acm.org/doi/10.1145/1459352.1459355 (visited on 06/02/2022).

[7] S. Joty, G. Carenini, R. Ng, and G. Murray, "Discourse Analysis and Its Applications", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 12–17. DOI: 10.18653/v1/P19-4003. [Online]. Available: https://aclanthology.org/P19-4003 (visited on 06/03/2022).

[8] H. Hernault, H. Prendinger, D. A. d. Verle, and M. Ishizuka, "HILDA: A Discourse Parser Using Support Vector Machine Classification", en, *Dialogue & Discourse*, vol. 1, no. 3, pp. 1–33, Dec. 2010, Number: 3, ISSN: 2152-9620. DOI: 10.5087/dad.2010.003. [Online]. Available: https://journals.uic.edu/ojs/index.php/dad/article/view/10670 (visited on 06/03/2022).

[9] A. Bilbao-Jayo and A. Almeida, "Automatic political discourse analysis with multi-scale convolutional neural networks and contextual data", en, *International Journal of Distributed Sensor Networks*, vol. 14, no. 11, p. 1 550 147 718 811 827, Nov. 2018, Publisher: SAGE Publications, ISSN: 1550-1329. DOI: 10.1177/1550147718811827. [Online]. Available: https://doi.org/10.1177/1550147718811827 (visited on 06/03/2022).

[10] S. M. Sarsam, H. Al-Samarraie, A. I. Alzahrani, and B. Wright, "Sarcasm detection using machine learning algorithms in Twitter: A systematic review", en, *International Journal of Market Research*, vol. 62, no. 5, pp. 578–598,

Sep. 2020, Publisher: SAGE Publications, ISSN: 1470-7853. DOI: 10.1177/1470785320921779. [Online]. Available: https://doi.org/10.1177/1470785320921779 (visited on 06/03/2022).

[11] S. Sun, C. Luo, and J. Chen, "A review of natural language processing techniques for opinion mining systems", en, *Information Fusion*, vol. 36, pp. 10–25, Jul. 2017, ISSN: 1566-2535. DOI: 10.1016/j.inffus.2016.10.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253516301117 (visited on 05/31/2022).

[12] F. Zhang, H. Fleyeh, X. Wang, and M. Lu, "Construction site accident analysis using text mining and natural language processing techniques", en, *Automation in Construction*, vol. 99, pp. 238–248, Mar. 2019, ISSN: 0926-5805. DOI: 10.1016/j.autcon.2018.12.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580518306137 (visited on 05/31/2022).

[13] "Data Mining", in *Mining of Massive Datasets*, A. Rajaraman and J. D. Ullman, Eds., Cambridge: Cambridge University Press, 2011, pp. 1–17, ISBN: 978-1-107-73741-9. DOI: 10.1017/CBO9781139058452.002. [Online]. Available: https://www.cambridge.org/core/books/mining-of-massive-datasets/data-mining/E5BFF4C1DD5A1FB946D616D619B373C2 (visited on 08/02/2022).

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", arXiv, Tech. Rep. arXiv:1810.04805, May 2019, arXiv:1810.04805 [cs] version: 2 type: article. DOI: 10.48550/arXiv.1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805 (visited on 06/03/2022).

[15] I. Tenney, D. Das, and E. Pavlick, “BERT Rediscovers the Classical NLP Pipeline”, arXiv, Tech. Rep. arXiv:1905.05950, Aug. 2019, arXiv:1905.05950 [cs] type: article. DOI: 10.48550/arXiv.1905.05950. [Online]. Available: http://arxiv.org/abs/1905.05950 (visited on 06/03/2022).

[16] M. V. Koroteev, “BERT: A Review of Applications in Natural Language Processing and Understanding”, arXiv, Tech. Rep. arXiv:2103.11943, Mar. 2021, arXiv:2103.11943 [cs] type: article. DOI: 10.48550/arXiv.2103.11943. [Online]. Available: http://arxiv.org/abs/2103.11943 (visited on 06/01/2022).

[17] A. A. Deshmukh and U. Sethi, “IR-BERT: Leveraging BERT for Semantic Search in Background Linking for News Articles”, arXiv, Tech. Rep. arXiv:2007.12603, Jul. 2020, arXiv:2007.12603 [cs] type: article. [Online]. Available: http://arxiv.org/abs/2007.12603 (visited on 06/06/2022).

[18] S. Edunov, A. Baevski, and M. Auli, “Pre-trained Language Model Representations for Language Generation”, arXiv, Tech. Rep. arXiv:1903.09722, Apr. 2019, arXiv:1903.09722 [cs] type: article. [Online]. Available: http://arxiv.org/abs/1903.09722 (visited on 06/06/2022).

[19] B. Lutkevich, *What is BERT (Language Model) and How Does It Work?*, en. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model (visited on 06/06/2022).

[20] S. Wu and M. Dredze, “Are All Languages Created Equal in Multilingual BERT?”, arXiv, Tech. Rep. arXiv:2005.09093, Sep. 2020, arXiv:2005.09093 [cs] type: article. DOI: 10.48550/arXiv.2005.09093. [Online]. Available: http://arxiv.org/abs/2005.09093 (visited on 06/07/2022).

[21] A. Virtanen, J. Kanerva, R. Ilo, *et al.*, “Multilingual is not enough: BERT for Finnish”, *arXiv:1912.07076 [cs]*, Dec. 2019, arXiv: 1912.07076. [Online]. Available: http://arxiv.org/abs/1912.07076 (visited on 02/15/2022).

[22] *TurkuNLP*. [Online]. Available: https://turkunlp.org/ (visited on 06/07/2022).

[23] *Common Crawl*, en-US. [Online]. Available: https://commoncrawl.org/ (visited on 06/07/2022).

[24] T. Seymour, D. Frantsvog, and S. Kumar, "History Of Search Engines", en, *International Journal of Management & Information Systems (IJMIS)*, vol. 15, no. 4, pp. 47–58, Sep. 2011, Number: 4, ISSN: 2157-9628. DOI: 10.19030/ijmis.v15i4.5799. [Online]. Available: https://www.clutejournals.com (visited on 02/14/2022).

[25] *Definition of Archie*, en. [Online]. Available: https://www.pcmag.com/encyclopedia/term/archie (visited on 04/01/2022).

[26] S. Hollingsworth, *DuckDuckGo vs. Google: An In-Depth Search Engine Comparison*, en, May 2021. [Online]. Available: https://www.searchenginejournal.com/google-vs-duckduckgo/301997/ (visited on 04/02/2022).

[27] D. Turnbull and J. Berryman, *Relevant search: With applications for Solr and Elasticsearch*. Manning Publications Co., 2016.

[28] *DB-Engines Ranking*, en. [Online]. Available: https://db-engines.com/en/ranking/search+engine (visited on 04/26/2022).

[29] *What Is Splunk? A Beginners Guide To Understanding Splunk*, en-US, Section: Big Data, Oct. 2016. [Online]. Available: https://www.edureka.co/blog/what-is-splunk/ (visited on 04/27/2022).

[30] *Who we are*. [Online]. Available: https://solr.apache.org/whoweare.html (visited on 04/27/2022).

[31] R. Kuć, *Solr vs Elasticsearch: Performance Differences & More [2022]*, en-US, Jan. 2022. [Online]. Available: https://sematext.com/blog/solr-vs-elasticsearch-differences/ (visited on 04/27/2022).

[32] *FAQ on 2021 License Change*, en-us. [Online]. Available: https://www.elastic.co/pricing/faq/licensing#faq-on-2021-license-change (visited on 04/28/2022).

[33] *Solr vs. Elasticsearch: Who's The Leading Open Source Search Engine?*, en-US, Jul. 2020. [Online]. Available: https://logz.io/blog/solr-vs-elasticsearch/ (visited on 04/28/2022).

[34] J. Blasenak, *Solr vs. Elasticsearch | Open Source Search | Accenture*, en. [Online]. Available: https://www.accenture.com/us-en/blogs/search-and-content-analytics-blog/solr-elasticsearch-open-source-search-engines (visited on 04/21/2022).

[35] *Schemaless Mode | Apache Solr Reference Guide 7.0*. [Online]. Available: https://solr.apache.org/guide/7_0/schemaless-mode.html (visited on 04/28/2022).

[36] *NearRealtimeSearch - Apache Lucene (Java) - Apache Software Foundation*. [Online]. Available: https://cwiki.apache.org/confluence/display/lucene/NearRealtimeSearch (visited on 05/02/2022).

[37] T. Aydoğan, M. İlkuçar, and M. A. Akca, "An Analysis on the Comparison of the Performance and Configuration Features of Big Data Tools Solr and Elasticsearch", en, *International Journal of Intelligent Systems and Applications in Engineering*, vol. 4, no. Special Issue-1, pp. 8–12, Dec. 2016, ISSN: 2147-6799. DOI: 10.18201/ijisae.271328. [Online]. Available: https://dergipark.org.tr/en/doi/10.18201/ijisae.271328 (visited on 04/27/2022).

[38] *Data in: Documents and indices | Elasticsearch Guide [master] | Elastic*, en-us, Learn/Docs/Elasticsearch/Reference/master. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/master/documents-indices.html (visited on 05/02/2022).

[39] A. Kryzhanovska and M. Sharapova, *How to Make a Search Engine Software for Your Business*, en. [Online]. Available: https://gearheart.io/articles/how-create-search-engine-software-your-business/ (visited on 04/12/2022).

[40] M. White, *Enterprise Search Development: Start With the User Interface*, en. [Online]. Available: https://www.reworked.co/information-management/enterprise-search-development-start-with-the-user-interface/ (visited on 05/12/2022).

[41] *Over 50 Percent of Knowledge Workers Cannot Find the Information They Need at Work, National Survey Finds*, en, Oct. 2019. [Online]. Available: https://www.businesswire.com/news/home/20191009005164/en/Over-50-Percent-of-Knowledge-Workers-Cannot-Find-the-Information-They-Need-at-Work-National-Survey-Finds (visited on 05/13/2022).

[42] S. Verma, *Designing for enterprise search*, en, Dec. 2021. [Online]. Available: https://uxdesign.cc/designing-for-enterprise-search-42015a9a467b (visited on 05/13/2022).

[43] S. Minhas, *How to Write Good Error Messages*, en, May 2021. [Online]. Available: https://uxplanet.org/how-to-write-good-error-messages-858e4551cd4 (visited on 05/13/2022).

[44] *Semantics | Definition & Theories | Britannica*, en. [Online]. Available: https://www.britannica.com/science/semantics (visited on 05/16/2022).

[45] S. Löbner, *Understanding Semantics* (Understanding language series). Routledge, 2013, ISBN: 9781444122435.

[46] H. Bast, B. Buchhold, and E. Haussmann, "Semantic Search on Text and Knowledge Bases", en, *Foundations and Trends® in Information Retrieval*, vol. 10, no. 1, pp. 119–271, 2016, ISSN: 1554-0669, 1554-0677. DOI: 10.1561/

1500000032. [Online]. Available: http://www.nowpublishers.com/article/Details/INR-032 (visited on 05/16/2022).

[47] S. Hosseinzadeh, S. Rauti, S. Laurén, *et al.*, "Diversification and obfuscation techniques for software security: A systematic literature review", en, *Information and Software Technology*, vol. 104, pp. 72–93, Dec. 2018, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2018.07.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584918301484 (visited on 05/24/2022).

[48] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "CodeSearchNet Challenge: Evaluating the State of Semantic Code Search", arXiv, Tech. Rep. arXiv:1909.09436, Jun. 2020, arXiv:1909.09436 [cs, stat] type: article. DOI: 10.48550/arXiv.1909.09436. [Online]. Available: http://arxiv.org/abs/1909.09436 (visited on 06/08/2022).

[49] X. Ye, J. Du, X. Gong, S. Na, W. Li, and S. Kudva, "Geospatial and Semantic Mapping Platform for Massive COVID-19 Scientific Publication Search", en, *Journal of Geovisualization and Spatial Analysis*, vol. 5, no. 1, p. 5, Jan. 2021, ISSN: 2509-8829. DOI: 10.1007/s41651-021-00073-y. [Online]. Available: https://doi.org/10.1007/s41651-021-00073-y (visited on 06/08/2022).

[50] J. Rygl, J. Pomikálek, R. Řehůřek, M. Růžička, V. Novotný, and P. Sojka, "Semantic Vector Encoding and Similarity Search Using Fulltext Search Engines", arXiv, Tech. Rep. arXiv:1706.00957, Jun. 2017, arXiv:1706.00957 [cs] type: article. [Online]. Available: http://arxiv.org/abs/1706.00957 (visited on 06/08/2022).

[51] T. Artyom, "Information retrieval and analysis for a Modern organization",, vol. 28, no. 4, pp. 7–28, 2016, ISSN: 2079-8156. [Online]. Available: https:

//cyberleninka.ru/article/n/information-retrieval-and-analysis-for-a-modern-organization (visited on 06/08/2022).

[52] H. N. L. Huy, H. H. Minh, T. N. Van, and H. N. Van, "Keyphrase Extraction Model: A New Design and Application on Tourism Information", en, *Informatica*, vol. 45, no. 4, Dec. 2021, ISSN: 1854-3871. DOI: 10.31449/inf.v45i4.3493. [Online]. Available: https://informatica.si/index.php/informatica/article/view/3493 (visited on 06/08/2022).

[53] A. Wen, Y. Wang, V. C. Kaggal, S. Liu, H. Liu, and J. Fan, "Enhancing Clinical Information Retrieval through Context-Aware Queries and Indices", in *2019 IEEE International Conference on Big Data (Big Data)*, Dec. 2019, pp. 2800–2807. DOI: 10.1109/BigData47090.2019.9006241.

[54] M. A. Silva-Fuentes, H. D. Calderon-Vilca, E. F. Calderon-Vilca, and F. C. Cárdenas-Mariño, "Semantic Search System using Word Embeddings for query expansion", in *2019 IEEE PES Innovative Smart Grid Technologies Conference - Latin America (ISGT Latin America)*, ISSN: 2643-8798, Sep. 2019, pp. 1–6. DOI: 10.1109/ISGT-LA.2019.8894992.

[55] I. Ö. Arnarsson, O. Frost, E. Gustavsson, M. Jirstrand, and J. Malmqvist, "Natural language processing methods for knowledge management—Applying document clustering for fast search and grouping of engineering documents", en, *Concurrent Engineering*, vol. 29, no. 2, pp. 142–152, Jun. 2021, Publisher: SAGE Publications Ltd STM, ISSN: 1063-293X. DOI: 10.1177/1063293X20982973. [Online]. Available: https://doi.org/10.1177/1063293X20982973 (visited on 06/08/2022).

[56] P. Kostakos, "Strings and Things: A Semantic Search Engine for news quotes using Named Entity Recognition", in *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*,

ISSN: 2473-991X, Dec. 2020, pp. 835–839. DOI: 10.1109/ASONAM49781.2020.9381383.

[57] S. V. Pakhomov, G. Finley, R. McEwan, Y. Wang, and G. B. Melton, "Corpus domain effects on distributional semantic modeling of medical terms", *Bioinformatics*, vol. 32, no. 23, pp. 3635–3644, Dec. 2016, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw529. [Online]. Available: https://doi.org/10.1093/bioinformatics/btw529 (visited on 07/13/2022).

[58] *Joutsenet käyttöliittymäkehityksen esiselvitys - loppuraportti*, ATR Soft Oy, Nov. 2020.

[59] *Kuntasektorin asianhallinnan viitearkkitehtuuri*, Kuntaliitto, 2016.

[60] *Home - Docker*, en-US, May 2022. [Online]. Available: https://www.docker.com/ (visited on 07/21/2022).

[61] *Quick start | Elasticsearch Guide [7.17] | Elastic*, en-us, Learn/Docs/Elasticsearch/Reference/7.17. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/7.17/getting-started.html (visited on 07/21/2022).

[62] *Text type family | Elasticsearch Guide [7.17] | Elastic*, en-us, Learn/Docs/Elasticsearch/Reference/7.17. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/7.17/text.html (visited on 07/21/2022).

[63] *Python Elasticsearch Client — Elasticsearch 7.17.4 documentation.* [Online]. Available: https://elasticsearch-py.readthedocs.io/en/v7.17.4/ (visited on 07/22/2022).

[64] J. Singer-Vine, *Pdfplumber*, original-date: 2015-08-24T03:14:48Z, Jul. 2022. [Online]. Available: https://github.com/jsvine/pdfplumber (visited on 07/21/2022).

[65] *Query DSL | Elasticsearch Guide [7.17] | Elastic*, en-us, Learn/Docs/Elasticsearch/Reference/7.17. [Online]. Available: `https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-dsl.html` (visited on 07/25/2022).

[66] *Elastic 8.0: A new era of speed, scale, relevance, and simplicity*, en-us. [Online]. Available: `https://www.elastic.coen-us/blog/whats-new-elastic-8-0-0` (visited on 07/25/2022).

[67] *Text similarity search in Elasticsearch using vector fields*, en-us, Aug. 2019. [Online]. Available: `https://www.elastic.co/blog/text-similarity-search-with-vectors-in-elasticsearch` (visited on 07/26/2022).

[68] *Home*, en. [Online]. Available: `https://elastiknn.com/` (visited on 07/26/2022).

[69] *TurkuNLP/sbert-cased-finnish-paraphrase · Hugging Face*. [Online]. Available: `https://huggingface.co/TurkuNLP/sbert-cased-finnish-paraphrase` (visited on 07/26/2022).

[70] *SentenceTransformers Documentation — Sentence-Transformers documentation*. [Online]. Available: `https://www.sbert.net/` (visited on 07/26/2022).

[71] S. Palachy, *Document Embedding Techniques*, en, Jun. 2022. [Online]. Available: `https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d` (visited on 07/27/2022).

[72] S. Verma, *Semantic Search with S-BERT is all you need*, en, Apr. 2022. [Online]. Available: `https://medium.com/mlearning-ai/semantic-search-with-s-bert-is-all-you-need-951bc710e160` (visited on 07/27/2022).

[73] M. Arora, U. Kanjilal, and D. Varshney, “Evaluation of information retrieval: Precision and recall”, *International Journal of Indian Culture and Business Management*, vol. 12, p. 224, Jan. 2016. DOI: `10.1504/IJICBM.2016.074482`.

[74] *Using the State Hook – React*, en. [Online]. Available: https://reactjs.org/docs/hooks-state.html (visited on 07/28/2022).

[75] *React Table*. [Online]. Available: https://react-table.tanstack.com/ (visited on 07/28/2022).

[76] styled-components, *Styled-components*, en. [Online]. Available: https://www.styled-components.com (visited on 07/28/2022).

[77] I. Beltagy, M. E. Peters, and A. Cohan, *Longformer: The Long-Document Transformer*, arXiv:2004.05150 [cs], Dec. 2020. DOI: 10.48550/arXiv.2004.05150. [Online]. Available: http://arxiv.org/abs/2004.05150 (visited on 10/19/2022).

[78] *Sentence Transformers: Multilingual Sentence, Paragraph, and Image Embeddings using BERT & Co.* original-date: 2019-07-24T10:53:51Z, Aug. 2022. [Online]. Available: https://github.com/UKPLab/sentence-transformers/blob/b9db7b0255d183565926159d2b1c16e67a969cf8/examples/applications/text-summarization/text-summarization.py (visited on 08/03/2022).

Appendix A Query Examples

A.1 The Main Query

The following request is an example of the main query, which is used to retrieve the search results to the search result table in the JoutseNet semantic search user interface. In the application, most indexed fields are listed in the fields arrays so that the query string is used to search from all of them, but it is not necessary to show the long list of fields in this example. To enable flexibility for the query string input, phrase_prefix and best_fields type queries are both utilized. Phrase_prefix accepts transposed terms with its slop parameter and best_fields accepts some character changes with its fuzziness parameter. Lastly, the highlight definition tells how to highlight certain fields if exact matches are found in them.

```

request = {
    "size": 300,
    "query": {
        "function_score": {
            "query": {
            "bool":{
                "should": [
                    {
                        "multi_match": {
                            "query" : query,
                            "type": "phrase_prefix",
                            "boost": 2,
                            "slop": 3,
                            "fields": [ "case_title",
                                        "textref_title",
                                        "file_content"]
                        }
                    },
                    {
                        "multi_match": {
                            "query": query,
                            "boost": 1.2,
                            "type": "best_fields",
                            "fuzziness": "AUTO:4,7",
                            "fields": [ "case_title",
                                        "textref_title",
                                        "file_content"]
                        }
                    },
                    {
                        "elastiknn_nearest_neighbors": {
                            "field": "vector_field",
                            "vec": {
                            "values": query_vector
                            },
                            "model": "lsh",
                            "similarity": "cosine",
                            "candidates": 100000
                        }
                    },
                ],
                "minimum_should_match": 1,
                "must": [
                    # User-defined filters are added to this 'must'
    array
                ],
                "filter": [
                    {"terms": {"user_group": user_group}}
                ]
                },
            },
        },
    },
    "highlight" : {
        "fragment_size" : 30,
        "order" : "score",
        "fields" : {
            "case_title" : { "number_of_fragments" : 2 },
            "textref_title" : { "number_of_fragments" : 2 },
            "file_content" : { "number_of_fragments" : 3 }
        }
    }
}
```

A.2 Example of a Filter Query

The following function is for implementing the dropdown list of suggestions for the author filter. Other filters with text input and suggestions are implemented with the same structure. The suggestions are aggregated with the aggs parameter so that there are no duplicates in the dropdown list. Flexibility for the input is enabled with the wildcard notations (.*). The query string is converted to lowercase because the aggregatable keywords are all in lowercase and the keyword search is case sensitive.

```

def _build_search_author_request(self, query):
    include = ".*" + query.lower() + ".*"
    return {
        "query": {
            "bool": {
                "must": {
                    "multi_match": {
                        "query" : query,
                        "type": "phrase_prefix",
                        "slop": 3,
                        "fields": ["author"]
                    }
                }
            }
        },
        "aggs": {
            "unique_authors": {
                "terms": {
                    "field": "author.raw",
                    "size": 100,
                    "include": include
                }
            }
        }
    }
```

The following function is the function for defining the author filter when an author is selected from the dropdown list of suggestions. The filter is appended to the must array of the main query: The search results not matching the filter conditions are filtered out of the search results.

```

def author_filter(author):
    return {
        "match": {
            "author.raw": author
        }
    }
```

Appendix B Document Embedding

There are two implementation suggestions for the document embedding in this appendix. The first one in appendix B.1 summarizes a parsed document and embeds the summary. If the document length is shorter than the defined summary length, the whole document is embedded. Document length is defined in the number of sentences. This implementation is based on the suggested technique presented in the SentenceTransformers documentation. [78] The document is split to sentences. Then each sentence is embedded, and cosine similarity is computed for all sentence pairs. Most central sentences are defined with the help of LexRank-algorithm and the summary is formed from those sentences.

The second implementation suggestion in appendix B.2 takes the parsed document text, splits it into smaller chunks, embeds the chunks and calculates the mean vector of the chunk embeddings. Then it returns the mean vector for indexing.

B.1 Embedding the Summary of a Document

```python
from sentence_transformers import SentenceTransformer, util
import nltk
import LexRank
import numpy as np

model = SentenceTransformer('TurkuNLP/sbert-cased-finnish-paraphrase')

def on_embed_file_content(text):
    prefixes = (' ', '.', '-')
    try:
        if text is not None:
            summary = ''
            text_without_digits = (''.join([i for i in text if not i.isdigit()])).replace('\n', '. ')
            sentences = nltk.sent_tokenize(text_without_digits, language='finnish')
            cleaned_sentences = [i for i in sentences if not (i.startswith(prefixes) or i == '')]
            if cleaned_sentences and len(cleaned_sentences) > 10:
                embeddings = model.encode(cleaned_sentences, batch_size=1)
                cos_scores = util.cos_sim(embeddings, embeddings).numpy()
                centrality_scores = LexRank.degree_centrality_scores(cos_scores, threshold=None)
                most_central_sentence_indices = np.argsort(-centrality_scores)
                for idx in most_central_sentence_indices[0:10]:
                    summary += (cleaned_sentences[idx])
            else:
                summary = text_without_digits
            summary_vector = model.encode(summary)
            return summary_vector
        else:
            text = ''
            text_vector =  [item for sublist in model.encode([text]) for item in sublist]
            return text_vector
    except Exception as ex:
        print(ex)
        embedding_error_logger.error("Exception occurred in document embedding", exc_info=True)
```

B.2 Embedding a Document in Smaller Chunks and Calculating the Average Vector

```python
from sentence_transformers import SentenceTransformer
import numpy as np
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('TurkuNLP/sbert-cased-finnish-paraphrase')
model = SentenceTransformer('TurkuNLP/sbert-cased-finnish-paraphrase')
model.max_seq_length = 510

def on_embed_file_content(text):
    try:
        if text is not None:
            text_without_digits = ''.join([i for i in text if not i.isdigit()])
            tokenized_text = tokenizer.tokenize(text_without_digits)
            token_chunks = [tokenized_text[i:i + 510] for i in range(0, len(tokenized_text), 510)]
            untokenized_chunks = [tokenizer.convert_tokens_to_string(i) for i in token_chunks]
            if untokenized_chunks:
                vectors = model.encode(untokenized_chunks, batch_size=1)
                vectors_mean = np.mean(vectors, axis=0)
                return vectors_mean
            else:
                text = ''
                text_vector =  [item for sublist in model.encode([
   text]) for item in sublist]
                return text_vector
        else:
            text = ''
            text_vector =  [item for sublist in model.encode([text
   ]) for item in sublist]
            return text_vector
    except Exception as ex:
        print(ex)
        embedding_error_logger.error("Exception occurred in
   document embedding", exc_info=True)
```