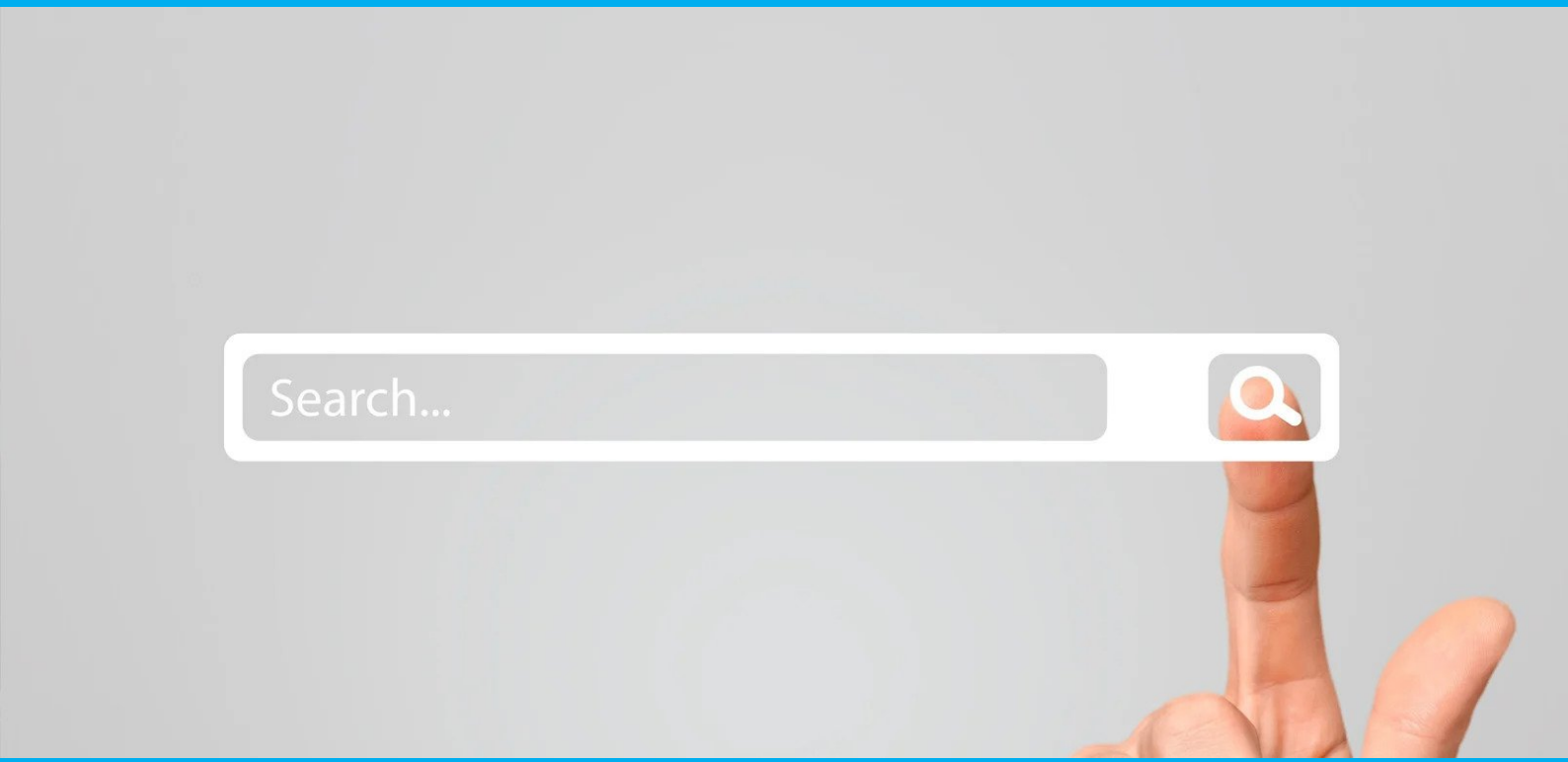
Rotterdam Werkt!

Improving interorganizational mobility through centralizing vacancies and resumes

Authors
L.E. van Hal
H.A.B. Janse
D.R. den Ouden
R.H. Piepenbrink
C.S. Willekens



TU Delft

Rotterdam Werkt!

Improving interorganizational mobility through centralizing vacancies and resumes

To obtain the degree of Bachelor of Science at the Delft University of Technology, to be presented and defended publicly on Friday January 29, 2021 at 14:30 AM.

Authors:	L.E. van Hal	
	H.A.B. Janse	
	D.R. den Ouden	
	R.H. Piepenbrink	
	C.S. Willekens	
Project duration:	November 9, 2020 – January 29, 2021	
Guiding Committee:	H. Bolk	*Rotterdam Werkt!*, Client
	R. Rotmans,	*Rotterdam Werkt!*, Client
	Dr. C. Hauff,	TU Delft, Coach
	Ir. T.A.R. Overklift Vaupel Klein	TU Delft, Bachelor Project Coordinator

TU Delft

Preface

This report denotes the end of the bachelor in Computer Science and Engineering at the Delft University of Technology. The report demonstrates all the skills we have learned during the bachelor courses in order to be a successful computer scientist or engineer. It will discuss the product we created over the past 10 weeks and will touch upon the skills and knowledge used in order to create it. Our client, *Rotterdam Werkt!*, is a network of companies located in Rotterdam who are aiming to increase mobility between their organizations. The goal of this report is to inform the reader about the complete work, different phases of this projects and future recommendations for our client.

Summary

Rotterdam Werkt! is a network of fourteen organizations in the Rotterdam area in the Netherlands. Their goal is to increase labor mobility between these organizations through sharing vacancies, exchanging employees and partaking in joint projects. *Rotterdam Werkt!* has tasked us with creating a central platform on which all vacancies are automatically combined from the websites of all the organizations in the network.

The two main challenges of the project were to gather the vacancies from all the organizations affiliated with *Rotterdam Werkt!* and allow their recruiters to search and filter through them. This meant that a significant amount of research needed to be done in order to find a suitable scraping tool as well as a suitable search engine. Whilst gathering the vacancies, we ran into the problem that each website was significantly different in the way it is rendered. Furthermore, we also needed to categorize the data correctly such that it becomes searchable in the search engine. Lastly, the retrieval function needed to be optimized such that the most relevant vacancies would be returned for a given query.

In order to assess whether recruiters could use the search engine effectively in practice, an evaluation of the effectiveness of the search engine was done. Three retrieval functions were compared based on a significance test of several effectiveness measures that indicate to what extent a retrieval function is able to retrieve relevant documents, or in this case, vacancies. Out of the three, the retrieval function that scored the highest was chosen to be used in the platform, so that recruiters will have a bigger chance to find the vacancies they will be looking for.

In the end, we consider our project to be a success. We managed to scrape all vacancies from all the websites of the organizations in *Rotterdam Werkt!* and to combine these on a centralized platform. Furthermore, the search engine evaluation allowed us to select the best vacancy retrieval function out of the three evaluated retrieval functions. However, more work can still be put into evaluating the search engine in the future by testing more retrieval functions based on more vacancy data, so that the search functionality can be further improved.

Acknowledgements

Before laying out the technical aspects of the final product, we would like to thank the people who have made this project possible. First of all, we would like to thank *Rotterdam Werkt!* for the opportunity to work on this project and provide us with the experience of working with real clients. We want to express our gratitude to Renée Rotmans, Henk Bolk and the recruiters for taking the time to meet with us to discuss the project and the direction we should be taking. Moreover, we would like to thank Claudia Hauff from the research group 'Web Information Systems' at the TU Delft for her guidance and insights into challenging concepts.

List of tables

2.1 Overview selected scraping libraries and frameworks . . . 7
2.2 Feature comparison selected scraping frameworks and libraries . . . 8
2.3 Overview of programming languages . . . 9
2.4 Feature comparison search engine frameworks . . . 11
2.5 Search engine frameworks requirements . . . 12
2.6 Search engine frameworks scores . . . 12
2.7 Most popular server frameworks on GitHub . . . 13
2.8 Multi criteria analysis chosen frameworks . . . 13
2.9 Job search website features . . . 14
2.10 Requirements evaluation . . . 17
2.11 Preliminary schedule . . . 20

3.1 List of fields extracted from the vacancy web pages by the scraper . . . 22

5.1 Requirements evaluation . . . 35
5.2 Retrieval functions and their weights . . . 40
5.3 Retrieval function precision metrics . . . 41
5.4 Retrieval function significance tests . . . 42

F.1 Search topics with search queries . . . 63

List of figures

2.1 Vacancy listings website STC Group 4
2.2 Vacancy listings website Evides 5
2.3 Search page features existing websites 16

3.1 Overall component diagram 22
3.2 Database diagram 23

4.1 High-level UML of REST-server 28
4.2 Kibana display error 30
4.3 Query logging statistics 31

5.1 Screenshots of the final product 37
5.2 SIG general analysis 43
5.3 SIG duplication refactor candidates 43
5.4 SIG unit size refactor candidates 44
5.5 SIG unit complexity refactor candidates 45

List of code snippets

2.1 HTML source STC Group 4
2.2 HTML source Evides Chrome DevTools 5
2.3 HTML source Evides 6
E.1 Elasticsearch vacancy mapping 57
E.2 Elasticsearch resume mapping 59
G.1 Query logging: Query 1 64
G.2 Query logging: Query 2 65
G.3 Query logging: Query 3 67
G.4 Query logging: Query 4 69
G.5 Query logging: Query 5 70

Contents

Preface ii

Summary iii

Acknowledgements iv

List of tables v

List of figures vi

List of code snippets vii

1 Introduction 1

2 Research 2
2.1 Overview . . . 2
2.2 Problem definition and analysis . . . 2
2.2.1 Problem analysis . . . 2
2.2.2 Problem statement . . . 2
2.2.3 Research topics . . . 2
2.3 Web scraping . . . 3
2.3.1 Static vacancy pages . . . 3
2.3.2 Dynamic vacancy pages . . . 5
2.3.3 Overview of frameworks and libraries . . . 6
2.3.4 Comparative analysis of frameworks and libraries . . . 7
2.3.5 Overview of programming languages . . . 9
2.3.6 Conclusion . . . 9
2.4 Search engine . . . 9
2.4.1 Search engine selection . . . 9
2.5 Website frameworks and libraries . . . 12
2.6 Related work . . . 13
2.7 Final requirements . . . 17
2.8 Design goals . . . 18
2.8.1 Security and privacy . . . 18
2.8.2 Maintainability . . . 18
2.8.3 Ease of deployment . . . 18
2.9 Approach . . . 18
2.9.1 Development methodology . . . 18
2.9.2 Documentation . . . 18
2.9.3 Version control . . . 19
2.9.4 Static code analysis . . . 19
2.9.5 Risk analysis . . . 19
2.9.6 Planning . . . 19

3 Design 21
3.1 Overview . . . 21
3.2 Architecture . . . 21
3.3 Scraper . . . 22
3.4 Database design . . . 23
3.5 REST-server . . . 24
3.6 Front-end design . . . 25
3.6.1 Interaction with resumes and vacancies . . . 25
3.6.2 Searching for resumes and vacancies . . . 25

4 Implementation **26**
4.1 Overview . . . 26
4.2 Scraper implementation . . . 26
4.2.1 Static pages . . . 26
4.2.2 Dynamic pages . . . 26
4.2.3 Spider automation . . . 27
4.3 REST-server implementation . . . 27
4.3.1 API and serializers . . . 27
4.3.2 Data storage . . . 27
4.4 Front-end implementation . . . 29
4.4.1 Form implementation . . . 29
4.4.2 Search implementation . . . 29
4.5 Logging . . . 29
4.5.1 Back-end logging . . . 29
4.5.2 Query logging . . . 30
4.6 Docker implementation . . . 31
4.6.1 REST-server . . . 31
4.6.2 Front-end . . . 31
4.6.3 Scraper . . . 32
4.6.4 Security . . . 32
4.7 Testing . . . 32
4.7.1 Scraper testing . . . 32
4.7.2 REST-server testing . . . 33
4.7.3 Front-end testing . . . 33

5 Product evaluation **34**
5.1 Overview . . . 34
5.2 Product evaluation . . . 34
5.3 Design goal evaluation . . . 34
5.3.1 Security and privacy . . . 34
5.3.2 Maintainability . . . 34
5.3.3 Ease of deployment . . . 36
5.4 Search engine evaluation . . . 37
5.4.1 Approach . . . 37
5.4.2 Cost evaluation . . . 38
5.4.3 Effectiveness evaluation . . . 38
5.4.4 Limitations . . . 42
5.5 Software Improvement Group . . . 42
5.5.1 First submission . . . 42
5.6 Ethical implications . . . 45

6 Process evaluation **46**

7 Conclusion and future work **47**
7.1 Conclusion . . . 47
7.2 Future work and recommendations . . . 47
7.2.1 Search engine evaluation . . . 47
7.2.2 Query logging . . . 48
7.2.3 Scrapers . . . 48

Appendices **49**

A Original problem statement **51**
A.1 Project description . . . 51

B Info sheet **52**

C *Rotterdam Werkt!* organizations **54**

D *Rotterdam Werkt!* information sheet **55**

E Elasticsearch mappings **57**
E.1 Vacancy 57
E.2 Resume 59

F Search topics **61**

G Logged queries **64**

1 Introduction

Rotterdam Werkt![1] is a network of fourteen organizations in the Rotterdam area, designed to increase labor mobility between the organizations through sharing vacancies, exchanging employees and partaking in joint projects. Currently, *Rotterdam Werkt!* has no platform to conveniently bundle each connected organization's vacancies. Consequently, the recruiters of *Rotterdam Werkt!* need to resort to exchanging vacancies via email whenever a new possibility for interorganizational labor mobility arises within the *Rotterdam Werkt!* network. In other words, there is no central hub that the organizations can use to view each other's vacancies, leading to a lack of overview and, potentially, to the exchange of outdated information. To maximize the benefits of a labor mobility network like *Rotterdam Werkt!*, quick and direct access to all connected organizations' vacancies is essential.

Currently, the organizations in *Rotterdam Werkt!* post their vacancies on their own websites. The task at hand is to develop a central platform that gathers all organizations' vacancies, to combine and display the gathered vacancies on a single web page, and to make the gathered vacancies accessible to all the organizations. As the final product we expect to deliver a solution to the problem defined in section 2.2. This will lead to the following deliverables:

1. A scraping tool which is able to run on regular intervals and collect all the vacancies which are open on the associated companies' websites.
2. A front-end and back-end containing a search engine which will allow the users, that is, the recruiters of *Rotterdam Werkt!*, to find vacancies which may be of interest to them.
3. A thesis report which will outline the approach taken to solve the problem stated in section 2.2.

The remainder of this report is divided into six chapters. In chapter 2 the background of the problem will be discussed, and research will be done into related work and into potential tools that can be used. Afterwards, in chapter 3, the design of the final product will be discussed together with the architecture of the different components, including their interaction. Chapter 4 describes the implementation of the product and provides a technical descriptions and solutions to the problems faced. Next, in chapter 5, the final product will be evaluated with regards to the requirements, design goals and effectiveness of the search engine. In chapter 6, an evaluation of the process with the lessons learned will be discussed. Finally, in chapter 7 the report will conclude with a reflection, conclusion and future work.

[1] See appendix D for more information.

2

Research

2.1. Overview

The first two weeks of the project were dedicated to doing research. In this chapter, all aspects of the research phase have been compiled. We started by analyzing the problem at hand in section 2.2. After that, an investigation into related work was done where the features of related platforms are identified. We then specified two areas in which we need to gain more knowledge: scraping and search engines. In section 2.2.3 we have therefore derived two research topics in which we need to strengthen our knowledge to create a successful final product, these will be further discussed in sections 2.3 and 2.4 respectively. In section 2.5, options will be discussed regarding the possible framework to be used in order to connect these two components. Section 2.6 provides an overview of related work regarding vacancy platforms. The final requirements are listed in section 2.7, and the design goals are presented in section 2.8. Lastly, the approach is discussed in section 2.9.

2.2. Problem definition and analysis

2.2.1. Problem analysis

Currently, *Rotterdam Werkt!* shares vacancies between HR-managers and recruiters via email and the company websites of all the fourteen affiliated organizations, which has become increasingly time-consuming. Therefore, *Rotterdam Werkt!* is looking for a new tool in order to facilitate this process. This tool must automatically collect vacancies of the websites of their current organizations as well as provide the means to search through these vacancies. The full list of organizations which will be supported can be found in appendix C.

2.2.2. Problem statement

The challenge presented by *Rotterdam Werkt!* consists of two parts:

1. **Automatically collecting all open vacancies of the participating companies**
   The challenge here is to create a tool which will gather the vacancies of the *Rotterdam Werkt!* organizations. This tool must be able to obtain data from, and adapt to, the varying vacancy data sources of each member of the network, without modifying or adding components to the data sources. These gathered vacancies need to be updated on a regular basis in order to find new ones and remove the ones which are no longer available.

2. **Allow recruiters of the participating companies to search for relevant vacancies**
   The challenge here is to create a search tool for the HR-managers and recruiters which is optimized for their application and type of data. In other words, to optimize the retrieval function in such a way that the best fitting vacancies are returned.

2.2.3. Research topics

From the challenges presented in section 2.2, two research topics can be identified in which we need to strengthen our knowledge:

1. What is the most suitable tool or software for collecting vacancies?

2. What is the most suitable search engine framework for this project?

These topics will be further explored in this chapter. Section 2.3 and 2.4 will investigate the first and second research topics, respectively. Lastly, section 2.5 will look into the web framework for connecting these two components of this project.

2.3. Web scraping

In order to create the vacancy search engine, the vacancies of the *Rotterdam Werkt!* member companies need to be obtained and categorized. Currently, all members have their own websites where they list their vacancies. In this section, we investigate which method we will be using to acquire the vacancies.

One option would be to create a central platform with a single database and have all organizations upload their vacancies to that platform. This would involve having people at the *Rotterdam Werkt!* organizations manually copy-and-pasting the information and categorizing it. Alternatively, if one had (API) access to the database of each organization, the data could be obtained already labeled, provided that all the members label their vacancy data instead of combining all information into a blob of text. This would be beneficial for filtering the vacancies, see section 2.5. Unfortunately, neither of the options are viable for *Rotterdam Werkt!*. The first option would require *Rotterdam Werkt!* members to upload their vacancies to two platforms – their own platform and a new central platform – as members still need to use their own website to attract applicants outside of the network. Managing two platforms in this manner could easily introduce consistency issues. Another possibility for this first option would be to have the members upload their vacancies only to the central platform and then embed the *Rotterdam Werkt!* platform on their websites. However, since all companies have different wishes for what data is displayed and in what way, it is unrealistic to expect to be able to make a platform all fourteen organizations can agree on within ten weeks. The second option is also unrealistic, since we believe it would be infeasible for each of the fourteen members to be able to provide access to their databases within the given time frame.

Given these limitations, their websites themselves are therefore the most reliable sources of information available. The *Rotterdam Werkt!* vacancy websites, like most websites, use a markup language named Hyper Text Markup Language (HTML). It defines the main data and structure of a page. HTML can be used to define a web page by itself, but most pages also use JavaScript and CSS within HTML. JavaScript is used to make a page interactive and CSS is used to define the style of the different elements of the page. In order to retrieve the relevant vacancy information, it must be extracted from the HTML source. Fetching a web page, parsing its contents and extracting data from it is called web scraping [1]. The web pages of the *Rotterdam Werkt!* organizations can be divided into two categories: static and dynamic. For both, an example of their layout, underlying HTML source and challenges they present will be given in section 2.3.1 and section 2.3.2, respectively.

2.3.1. Static vacancy pages

An example of a static vacancy page is the overview page of STC Group[1], shown in figure 2.1. The general layout is representative of many vacancy overview pages within the *Rotterdam Werkt!* network. The vacancies are often presented in blocks, with one vacancy per block, or in a table. For the websites with a block-layout, such as the example of STC Group, the blocks often contain the title of the vacancy, a short description, and important job attributes, such as the hours, location or department. Most importantly, there is always some text or button with a hyperlink to the vacancy detail page. These hyperlinks must be extracted by the scraper and followed so the details of each vacancies can be scraped as well.

Code snippet 2.1 shows part of the source HTML for this web page. For example, one can see that the `div` elements with the `class` name `col-xl-12` are the source of the blocks with the vacancies. Within this `div`, there is relevant vacancy data to extract, such as the `h2` element that contains the title of the vacancy: “Technisch docent Zeevaart (HWTK)”. Most importantly, the anchor element, `<a>`, contains the location of the vacancy detail page we are looking for. In order to extract information such

[1] https://werkenbijstc.nl/vacatures/

as the vacancy title and hyperlinks, we therefore need a library that can parse HTML and extract the relevant data.

Figure 2.1: Vacancy listings on the STC Group website. Screenshot taken on 20 Jan. 2021.

```
<!DOCTYPE html>
<html lang="nl">
  ...
  <body>
    ...
    <section class="stc-vacancy--overview">
      <div class="container">
      <div class="row">
        <div class="col-xl-12">
          <div class="stc-vacancy--box featured">
            <div class="stc-vacancy--head">
              <h2 class="stc-vacancy--title">Technisch docent Zeevaart (HWTK)</h2>
              <span class="stc-vacancy--label">Onderwijs</span>
            </div>
            <div class="stc-vacancy--content">
              <p>
                Heb jij zelf gevaren en weet jij alles van de technische kant van schepen? Gaat jouw
                hart sneller kloppen van motoren, voorstuwing, onderhoud en hulpmotoren en weet je jouw
                passie met enthousiasme over te brengen op onze mbo studenten? Lees dan snel verder!
              </p>
            </div>
            <div class="stc-vacancy--footer">
              <div class="stc-vacancy--info">
                <span class="stc-vacancy--info-label">32-40 uur</span>
                <span class="stc-vacancy--info-label">Zwolle</span>
              </div>
              <div class="stc-vacancy--more">
                <a class="btn arrow-text-btn btn-secondary"
                   href="https://werkenbijstc.nl/vacatures/technisch-docent-zeevaart-hwtk/">
                   Bekijk vacature
                </a>
              </div>
            </div>
          </div>
          <div class="col-xl-12">
            ...
          </div>
        </div>
      </div>
    </section>
    ...
  </body>
</html>
```

Code snippet 2.1: Partial HTML source of the vacancy listings on the STC Group website. Only the elements that make up the first vacancy have been completely expanded. Source downloaded using `wget` on 20 Jan. 2021.

2.3.2. Dynamic vacancy pages

In addition to static vacancy pages, there are also dynamic ones. A dynamic web page, more specifically in this case a client-side dynamic web page, is a page that uses client-side scripting, such as JavaScript, to alter the Document Object Model (DOM) of a web page during or after it has loaded. In figure 2.2, the vacancy overview page of Evides[2] is shown. Looking at the layout, the site seems similar to the STC Group static example. This overview page uses a table instead of blocks, but the important information such as the title and location is still there. The hyperlinks seem missing, but are actually hidden within the rows of the table. Hovering over a row with a cursor will turn that row blue to indicate to a user that it is a clickable link leading to more information.

Looking at the HTML via a browser's developer tools, shown in code snippet 2.2, seems to confirm that the Evides and STC Group websites are also similar HTML-wise. This is not true, however. Using a tool such as `wget`[3] to download the original HTML source shows that the `a` elements with the `dataRow` class are not present. The vacancy data is instead hidden within the JavaScript defined inside the `<script>` elements, as shown in code snippet 2.3. The table will only appear after a browser has executed the JavaScript code. In order to extract this data, one must either use a JavaScript engine, such as browsers do, and then extract the data from the HTML the same way as for the static vacancy pages, or parse the JavaScript code itself.

Vacatures

Denk jij dat je bij ons snel komt bovendrijven vanwege je kennis, ervaring en persoonlijkheid?
Wil jij samen met vakkundige collega's bijdragen aan het creëren van waardevol water?

Dan geeft Evides jou alle mogelijkheden om een goede carrièrestart te maken of een volgende stap in je carrière te zetten.

Functie	Standplaats
Accountmanager Techniek	Kralingen, Rotterdam
Scrum Master	Kralingen, Rotterdam
Projectleider A - Operationele Ondersteuning	Kralingen, Rotterdam
Projectleider bouwkunde	Kralingen, Rotterdam en Middelburg, Zeeland

Figure 2.2: Vacancy listings on the Evides website. Screenshot taken on 20 Jan. 2021.

```
<!DOCTYPE html "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html id="P_html" xmlns="http://www.w3.org/1999/xhtml" class="responsive" lang="nl" xml:lang="nl">
  ...
  <body id="P_body" class="sitebackground desktop" style="overflow: auto scroll; visibility: visible;">
    ...
    <div class="body">
      <a class="dataRow odd" href="vacaturebeschrijving-2020/accountmanager-klant">
        <div class="dataColumn verticalCenter" style="width: 512px;">
          <div class="text" data-row-id="6595754c-f018481ab50d">Accountmanager Techniek</div>
        </div>
        <div class="dataColumn verticalCenter" style="width: 686px;">
          <div class="text" data-row-id="6595754c-f018481ab50d">Kralingen, Rotterdam</div>
        </div>
      </a>
      <a class="dataRow even" href="vacaturebeschrijving-2020/scrum-master">
        ...
      </a>
        ...
    </div>
    ...
  </body>
</html>
```

Code snippet 2.2: Partial HTML of the vacancy listings on the Evides website as shown in Chrome DevTools. Only the elements that make up the first vacancy have been completely expanded. Downloaded with Chrome on 20 Jan. 2021.

[2] https://www.werkenbijevides.nl/
[3] https://www.gnu.org/software/wget/

```
<!DOCTYPE html "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html id="P_html" xmlns="http://www.w3.org/1999/xhtml" class="responsive" lang="nl" xml:lang="nl">
  ...
  <body id="P_body" class="sitebackground" style="overflow-y:scroll;overflow-x:auto;visibility:hidden;">
    ...
    <script type="text/javascript">
        function AntaRequireBootstrapCompleted(AntaRequire) {
          AntaRequire(...) {
            script.AddInitScript(function () {
                ...
                webview.RegisterWebView("P_C_W_B2782DF749637890AA5FD88EFDA8A090", {
                    "data": [{
                        "uniqueId": "60c11015-b416-4901-9947-16e329c6b6a4",
                        "rowNumber": 1,
                        "link": "vacaturebeschrijving-2020/accountmanager-klant",
                        "columns": {
                          "U005": {
                            "value": "Accountmanager Techniek"
                          },
                          "U004": {
                            "value": "Kralingen, Rotterdam"
                          },
                          "KnWCITitl": {
                            "value": "Accountmanager Klant"
                          },
                          "KnWPgPuId": {
                            "value": "605CD65342D26ACAD141D6A36DF36C3A"
                          },
                          "KnWCIDs": {
                            "value": "Accountmanager Klant"
                          },
                          "KnWCIPuDa": {
                            "value": "2020-11-05T13:39:06"
                          }
                        }
                      },
                      ...
                    });
                });
            });
        };
    </script>
    ...
  </body>
</html>
```

Code snippet 2.3: Partial HTML/JavaScript source of the vacancy listings on the Evides website. Only the elements that make up the first vacancy have been completely expanded. Source downloaded using `wget` on 20 Jan. 2021.

2.3.3. Overview of frameworks and libraries

A selection of four distinct scraping libraries and frameworks has been made based on the following three criteria:

- Gratuitous (libre)
- Open-source
- Popularity

Given the lack of funds, the framework or library of choice must be free-to-use, as will be further explained in section 2.7. As our product is to be used in a commercial setting, it must not be restricted by licenses or patents, so the selected software of choice must be open-source. Finally, the software must be used widely, because we believe that this increases the likelihood of having comprehensive documentation, fewer bugs and better support. The popularity of the frameworks and libraries has been determined by number of contributors, latest release date and the ranking in Google search results. An overview of the four candidates is shown in table 2.1.

	Beautiful Soup	Scrapy	Cheerio	libxml2
Information				
Source repository	Launchpad	GitHub	GitHub	GitLab
Latest release	8 Oct 2020	17 Nov 2020	21 Dec 2020	30 Oct 2019
Number of contributors	20	397	115	99
Documentation	Provided by developer	Provided by developer	Provided by developer	Provided by developer
Language				
Written in	Python	Python	JavaScript	C (also bindings for C++, Ruby, Python, PHP, etc.)

Table 2.1: Overview of the selected scraping libraries and frameworks [2]–[5].

Beautiful Soup
Beautiful Soup is a Python library that can extract data from HTML and XML documents. Given an HTML or XML document, it creates a parse tree from which relevant information can be extricated [2]. It can also process structurally incorrect HTML documents, such as those with malformed tags or improper nesting. The processing speed and tolerance for malformations depends on the parser used. Beautiful Soup supports Python's built-in `html.parser`, but also third-party parsers such as `lxml` and `html5lib` [6]–[8].

Scrapy
Scrapy is a scraping and web crawling framework written in Python. It is based around modules called *spiders*. These spiders are essentially web crawlers that follow the instructions given by the programmer in Python. They also serve to separate the web crawling aspect from the actual data extraction, which allows for easier maintainability [3].

Cheerio
Cheerio describes itself as a "Fast, flexible, and lean implementation of core jQuery designed specifically for the server" [9]. This JavaScript package can be used for web scraping purposes such as parsing and manipulating data from the HTML DOM, which can be obtained using an external request library.

libxml2
Another library that is considered is libxml2. This library can be used to parse and navigate XML or HTML [5]. Despite the fact that this library is written in C, libxml2 can be used in other environments as well, thanks to its available bindings [5].

2.3.4. Comparative analysis of frameworks and libraries

In order to make a justified decision on which framework or library to use for this project, the candidates ought to be compared and contrasted. This has been done based on a set of criteria we deem to be important for the development of the web scraper within the given time frame of ten weeks for this project. An overview of the supported features per scraping framework or library can be found in table 2.2.

First of all, web responses must be parsed to be able to extract data from them. Given that websites are written in HTML, a parser for that language is needed. Likewise, some of the client's websites, such as Evides' vacancy website, rely on JavaScript to dynamically load their content. This means that a JavaScript parser, or a similar feature, would be needed. If a website makes use of a JSON API, the data can be obtained directly from that API, rather than obtaining it from the actual user interface. A JSON parser is needed to get the data from the API response.

Secondly, data must be extracted from the parsed input. This can be done in various ways, for instance with CSS selectors, XPath expressions or RegEx.

Thirdly, for quality assurance, the scraper should be tested and debugging options would be beneficial for the development process. Hence, logging and automated testing features are taken into consideration.

Finally, miscellaneous features are also considered. Invalid scheme parsing and automatic request retry on error are included in this comparative analysis, as a situation could arise in which the scheme

is incomplete, for instance due to an invalid or missing HTML tag, or the connection to website could fail. Data sanitization and data export are also included, because the data needs to be exported to the server and said data must be clean; for instance, the data must not contain unreadable characters, like tabs or newlines. Asynchronous crawling is considered as this could prove to be useful in case scraping all websites sequentially might take a long time. Running the scraper asynchronously would allow it to scrape multiple websites at the same time, improving the run time of the scraper. Additionally, it allows the data to be processed while waiting for the other requests.

	Beautiful Soup	Scrapy	Cheerio	libxml2
Parsing				
HTML	●	●	●	●
XML	●	●	●	●
JavaScript	○	□	○	○
JSON	◐	◐	◐	◐
Data extraction				
CSS selectors	●	●	●	○
XPath	●	●	●	●
RegEx	●	○	○	○
Scraper debugging and testing				
Logging	●	●	○	○
Automated regression tests	○	◐	○	○
Miscellaneous features				
Invalid scheme parsing	●	●	●	●
Data sanitization	○	○	○	○
Data export	○	●	○	○
Automatic request retry on error	○	●	○	○
Asynchronous crawling	○	●	○	○

●= Supported; ○= Not supported; ◐= Available through extensions; □= Similar feature available

Table 2.2: Feature comparison of the selected scraping frameworks and libraries [2]–[5].

Based on the overview of the selected scraping libraries and frameworks in table 2.1 and feature comparison in table 2.2, several key aspects become clear.

Firstly, there is a distinction between frameworks and libraries. More specifically, this distinction is related to the *inversion of control* [10]. When using a library, developers control the flow of the program. When needed, they can call a library function which will execute its task and return control back to the developer, which allows for a higher degree of customizability. When using a framework, the framework decides when and what actions take place. This system allows for modularity and extensibility, but makes frameworks less customizable. Scrapy is the only framework in this comparison. It controls the flow of execution, only allowing developers to define callbacks. In exchange, Scrapy has many built-in features and allows for extensibility by means of middlewares[4]. Given the time constraint of this project, the provided features outweigh the lesser degree of control, as time can be saved by not having to implement the features Scrapy already has.

Secondly, data parsing seems to be supported by all candidates, though JavaScript seems to require additional attention. While Scrapy has a similar feature to support JavaScript by means of Scrapy Selenium[5], the other candidates do not. This means that more time is needed for the implementation for the other candidates.

Thirdly, data extraction does not pose an issue, as all candidates support XPath expressions. Even though additional data extraction methods would provide more programming flexibility, XPath expressions would suffice.

Fourthly, regarding debugging and testing features, Scrapy seems to be the superior choice, followed by Beautiful Soup. Scrapy has comprehensive logging features[6] and Beautiful Soup provides an on-parse debug feature[7]. If any of the other two candidates were chosen, extra time has to be spent on integrating testing and debugging features tailored to web scraping.

[4] https://docs.scrapy.org/en/latest/topics/spider-middleware.html
[5] https://github.com/clemfromspace/scrapy-selenium
[6] https://docs.scrapy.org/en/latest/topics/logging.html
[7] https://beautiful-soup-4.readthedocs.io/en/latest/#troubleshooting

Finally, the miscellaneous features. Most time can be saved by using Scrapy, as it has built-in data export features, supports asynchronous crawling and automatically retries a request when a request error occurs, while the others do not. None of the candidates has built-in data sanitization, which means that we have to implement this ourselves or through the use of external libraries.

2.3.5. Overview of programming languages

Table 2.3 lists the programming languages that are used by the libraries and framework discussed in section 2.3.4. When choosing a library or framework, the associated programming language should also be considered as experience with and development time using a given language may differ greatly.

	Python (Scrapy and Beautiful Soup)	JavaScript (Cheerio)	C (libxml2)
Language type	interpreted high-level	interpreted high-level	compiled low-level
Team's experience	High	Medium	Low

Table 2.3: Overview of programming languages [11]–[13].

Both Python and JavaScript are interpreted high-level languages [11], [13]. Unlike low-level languages, such as C, high-level languages support numerous features that we believe to greatly improve ease of development, such as memory management and garbage collection [14]. However, such languages are not recommended if maximum performance is required, due to the abstraction penalty of using a high-level language [14]. Given that run time is of little concern for this project and development time is limited, a high-level language is preferred. With the team's experience in mind, it becomes clear that Python is preferred over JavaScript. While the former can be used efficiently immediately, the latter requires more familiarization, and thus more time. As mentioned earlier, time is rather limited, so the main focus should be on just learning how to use a library or framework rather than a language as well.

2.3.6. Conclusion

In conclusion, while Beautiful Soup, Cheerio and libxml2 can be useful in a certain context, Scrapy seems to be the clear winner. Not only does Scrapy allow for the best basis for meeting the requirements, it also eases the development thanks to its scraping and supportive features. The fact that Scrapy is written in Python is also an asset, since the development time in Python is relatively short and numerous tools for post processing are available.

2.4. Search engine

After having retrieved all the vacancies (see section 2.3), a search engine is needed in order to be able to search and filter through the vacancies. This is necessary to meet the final product's requirement of giving users the ability to search for specific vacancies according to search queries. In this section, several existing search engines will be analyzed and their features will be compared. We will first indicate how search engines were selected for analysis, followed by an overview of each of the search engines' features. We conclude this section by arguing whether one of the selected search engines will be used for realizing the final product, or if we need to implement a custom search engine.

2.4.1. Search engine selection

There are several criteria that a search engine framework needs to adhere to for it to be considered in the creation of the final product. One is that the framework should (be able to) possess all the required features as presented in section 2.6. Furthermore, the framework should be reliable enough to be of any practical use. Lastly, budgetary constraints force us to use a low-cost and preferably free-to-use framework, which is why only open-source frameworks will be considered. Like in section 2.3.3, we select four frameworks for further review based on popularity, which we derive from a combination of Google search result ranking and user popularity on GitHub, and based on whether they are open-source. The frameworks that will be compared are Elasticsearch, Solr, Manticore Search and Xapian.

Feature comparison and analysis

For each of the selected search engine frameworks, major features were extracted from their main websites[8] . Table 2.4 shows an overview of the features of each of the frameworks. Comparing each framework's features, the following characteristics can be examined.

Documentation, actuality and popularity Each of the frameworks provides extensive documentation which explains the inner working of each framework and which instructs on how to use each framework practically. All frameworks have also had their newest release in June 2020 at the earliest, which indicates that they are all relatively up-to-date. However, if we assume popularity among users can be measured in terms of GitHub stars and forks, Elasticsearch is clearly the most popular of the four, followed by Solr, Xapian and Manticore Search, respectively.

Programming language compatibility All shown frameworks support a wide range of programming languages. Elasticsearch and Solr make use of RESTful interfaces[9] which are supported by many major development programming languages, including but not limited to Python, JavaScript and C++, which were already mentioned in section 2.3.5. Manticore Search provides the same support through MySQL connectors[10]. Xapian supports many programming languages as well through C++ bindings, but Xapian's Java API is still experimental [16].

Language features in search queries Elasticsearch and Manticore Search provide out-of-the-box word synonym searching and extensive Natural Language Processing (NLP) support, with features such as word tokenization and stemming, for the Dutch language. Xapian supports searching for synonyms as well, but only provides word stemming as a Dutch NLP-feature. Solr also has support for NLP and synonym search, but this requires the addition of certain extensions. All engines feature a form of spell correction for terms in search queries, based on words in the engines' databases, words from a corpus or a combination thereof.

Additional search features Apart from simply searching for terms in their databases, all frameworks provide additional operators that can be used to narrow down searches. For instance, all frameworks support boolean operators and wildcards in queries. Manticore Search provides the most out-of-the-box additional search features and filters, followed in order by Elasticsearch, Solr and Xapian.

Final selection criteria

The selection of a search engine framework for use in the creation of the final product depends on several factors. In section 2.4.1, we already selected frameworks for further analysis based on their presumed reliability, the expectation that they have the features that are requested for the final product, and their availability as an open-source framework. There are several more factors that need to be taken into consideration for the selection of a framework to be used in the creation of the final product.

Firstly, there is a time constraint of ten weeks to complete the product. We deem it to be impractical to develop our own search engine framework that can compare to the frameworks that were analyzed in section 2.4.1 in this time frame, and we will therefore be using one of the presented frameworks. Furthermore, considering this time frame, it is important that the selected framework has a relatively small learning curve, which depends on our expertise, and that there is extensive documentation, including any reference guides and tutorials that the framework developers publish, but also user contributions posted online elsewhere. In assessing to what extent the latter is likely to be present for each framework, we look at each framework's *stars*[11] and *forks*[12] on their GitHub repositories, which is a direct indicator of the size of the community of users that are in some way connected to a framework.

[8] Elasticsearch: https://www.elastic.co/; Solr: https://lucene.apache.org/solr/; Manticore Search: https://manticoresearch.com/; Xapian: https://xapian.org/.

[9] REST is an architectural style that defines certain constraints for interfaces, designed to make the web more standard. As such, RESTful interfaces are interfaces that adhere to these constraints [15].

[10] For more information on MySQL connectors, see https://dev.mysql.com/doc/connectors/en/.

[11] Number of times somebody followed this project. Usually indicates how interested people are in the project.

[12] Number of times the repository is copied, usually with the intent of making a change or addition to the code base. Usually indicates how actively updated the repository is.

Feature	Elasticsearch	Solr	Manticore Search	Xapian
Information				
GitHub repository	52.2k stars, 18.2k forks	3.9k stars, 2.6k forks	497 stars, 68 forks	601 stars, 264 forks
Latest release	01 June 20	02 Nov 20	01 Oct 20	21 Aug 20
Documentation	Provided by developer	Provided by developer	Provided by developer	Provided by developer
Language				
Written in	Java	Java	C++	C++
Supported programming languages	RESTful HTTP/JSON API, Java JavaScript, Perl, PHP, Python, Ruby	REST-interface-compatible languages	Any language with MySQL-compatibility	Bindings for Python, Java, Perl, PHP, Tcl, C#, Ruby, Lua, Erlang, Node.js, R
Database type	NoSQL	Built-in; input JSON, XML, CSV or binary; query with HTTP GET	SQL or JSON; SQL queries or HTTP requests	Built-in; based on CSV
Extensive NLP support for Dutch language	●	◐	●	Word stemming only
Spell correction	●	●	●	●
Synonym support	●	◐	●	●
Search features				
Ranked search	●	●	●	●
Faceted search	●	●	●	●
Proximity search	●	●	●	●
Fuzzy search	●	●	□	○
Range search	●	●	●	●
Attribute search	●	●	●	●
Auto-fill or term suggestions	●	●	●	□
Search operators				
Boolean	●	●	●	●
Wildcards	●	●	●	●
Quorum matching	○	○	●	○
Strict order	●	◐	●	○
Field-start/-end	○	○	●	○
Sentence-/paragraph-specific	○	○	●	○

●= Supported; ○= Not supported; ◐= Available through extensions; □= Similar feature available

Table 2.4: Search engine frameworks and a comparison of their features as listed on their main websites. [17]–[20]

It should also be taken into account that the final product is required to include several features as presented in section 2.7. Table 2.5 shows an overview of the required features and shows for each analyzed framework whether it supports the feature, based on the findings in table 2.4.

Summarizing, the criteria that will be used for selecting a search engine framework are the amount of documentation published by the developer; user popularity; learning curve; and support for required features.

Selection of a Search Engine Framework

In order to select the framework that will be used in the final product, we assess to what extent each framework meets the criteria described in section 2.4.1. This is measured based on the findings in table 2.4 and table 2.5. Learning curve was assessed based on our individual expertise of used database engines and connection and query methods, as well as perceived quality and user-friendliness of provided documentation. Table 2.6 shows an overview of each framework's scores per criterion.

As Xapian does not directly and fully provide support for all required features, Xapian is taken out of consideration. Furthermore, even though Solr has a relatively high user popularity, it has a higher learning curve than the other frameworks and it requires extensions for the essential features to be covered, which may prove impractical given the time frame in which the final product must be completed, and so Solr will also not be selected.

Feature	Elasticsearch	Solr	Manticore Search	Xapian
Features for employees				
Search by keywords	●	●	●	●
Search by function	●	●	●	●
Search by location	●	●	●	●
Search by company name	●	●	●	●
Filter categories	●	◐	●	●
Show similar vacancies	●	◐	●	□
Features for recruiters				
Search for CV's	●	◐	●	□
Non-functional features				
Easily maintainable and extendable code	●	●	●	□; Java API still experimental

●= Supported; ○= Not supported; ◐= Available, but not out-of-the-box; □= Similar feature available or partly supported

Table 2.5: Overview of search engine frameworks with required features in the final product

Criterion	Elasticsearch	Solr	Manticore Search	Xapian
Documentation	Provided by developer	Provided by developer	Provided by developer	Provided by developer
User popularity based on GitHub stars and forks	Highest	Second-highest	Lowest	Second-lowest
Learning curve (based on experience and expertise)	Low	Higher	Lowest	Low
Out-of-the-box support for required features	8/8	5/8	8/8	5/8
Support for required features with extensions	8/8	8/8	8/8	5/8

Table 2.6: Scores of search engine frameworks based on requirement criteria

In general, it is believed that Elasticsearch has a higher learning curve compared to Manticore Search, however there is a large gap in user popularity in favor of Elasticsearch [21]. This shows that Elasticsearch has a more active user base, and therefore more user-provided documentation compared to Manticore Search. This is also visible when we compare the amount of questions posted on Stack Overflow[13] that relate to Elasticsearch and those that relate to Manticore Search. To find such questions, we filter based on *tags*[14]. In total, 47445 questions are returned for Elasticsearch, versus 5 questions for Manticore Search (or, 39 if widening the search to any question containing "Manticore Search"), indeed indicating a much higher interest in Elasticsearch than in Manticore Search on Stack Overflow. Considering the large user base of Stack Overflow, we believe this to be at least somewhat representative of the true interest in the frameworks, and also of the available user-made documentation on the frameworks.

All factors considered, we therefore believe Elasticsearch will be the best framework in the application of the final product. Additionally, Elasticsearch is part of the ELK stack[15], which also contains a logging module, Logstash, and a data visualizer, Kibana. This means that we do not need to implement these features ourselves, thus save precious time.

2.5. Website frameworks and libraries

To run our application we will need a server framework and a front-end library. Several frameworks were chosen for evaluation based on their popularity on GitHub. Table 2.7 contains the server frameworks that were found. We are looking for a relatively simple framework that will allow us to easily implement our required features. Considering our vacancies will be stored in a database, we need a (RESTful)

[13] Stack Overflow is an online platform operating since 2008, on which 100 million people ask questions and share knowledge about coding problems monthly as of November 2020. See https://stackoverflow.com/. Accessed 12 November 2020.

[14] Users can connect descriptive tags to questions on Stack Overflow so that their questions can be easily found by anyone interested. We used the tag Elasticsearch to find questions about Elasticsearch, and manticore, manticoresearch and manticore-search to find questions about Manticore Search.

[15] https://www.elastic.co/what-is/elk-stack

API which exposes methods such as POST, PUT, GET and DELETE in order to access and mutate the data in the database [22].

Feature	Spring	Django	Ruby on Rails	Laravel
Information				
GitHub repository	40.1k stars, 27.6k forks	53.5k stars, 23.1k forks	46.9k stars, 18.8k forks	62.5k stars, 19.8k forks
Latest release	01 Jun 20	02 Nov 20	01 Oct 20	21 Aug 20
Documentation	Provided by developer	Provided by developer	Provided by developer	Provided by developer
Properties				
Language used	Java	Python	Ruby	PHP

Table 2.7: Current most popular server frameworks on GitHub on 15 Nov. 2020

Criterion	Spring	Django	Ruby on Rails	Laravel
User popularity based on GitHub stars and forks	Fourth	Second	Third	First
Learning curve (based on past experience)	Low	Low	High	High
Language experience	High	High	Low	Low

Table 2.8: Multi criteria analysis for the chosen frameworks

As shown in table 2.8, the highest scoring frameworks are Django, Spring and Laravel. Because most web scrapers are in Python, and we are most comfortable with Python, we will choose Django as our server-side framework.

For the front-end, we chose to use React because of the vast number of libraries available for it, including Search-UI[16]. Search-UI provides us with a component which makes interfacing with Elasticsearch easier. This significantly reduces the complexity and the need for manually creating a search box, a results page, implement paging and filters. Therefore, the choice for React was made on the basis that this library will significantly reduce the development time which is essential given the short duration of this project.

2.6. Related work

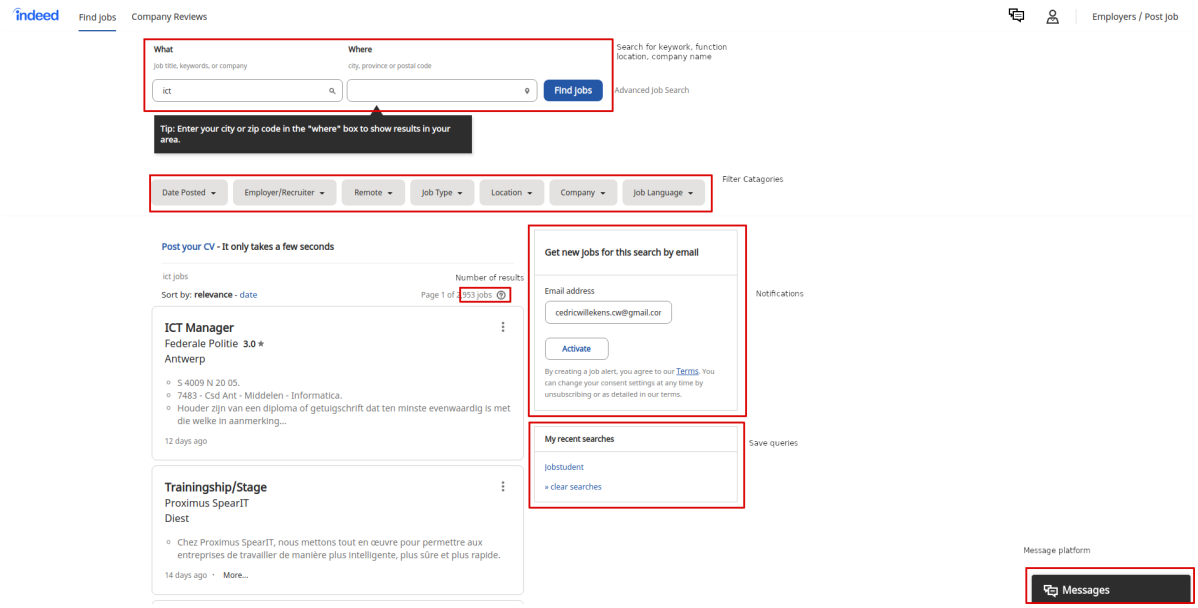
In order to better understand the possibilities of the already existing job sites, and because the client wishes to create '*Something like Indeed but private*', a comparison matrix was created containing all the features popular vacancy websites have. The matrix contains vacancy websites which showed up at the top of a Google web search. This allows us to select the most popular ones. A table with these features can be found in table 2.9.

Features which are used by three or more companies are believed to be worth investigating further. Therefore those features have been marked with an asterisk ($\star$) in table 2.9. These features will be presented to the client, which will be used together with their feedback to create the final requirements in section 2.7. In order to further clarify the features described in table 2.9, figure 2.3 contains screenshots from vacancy websites which include these features.

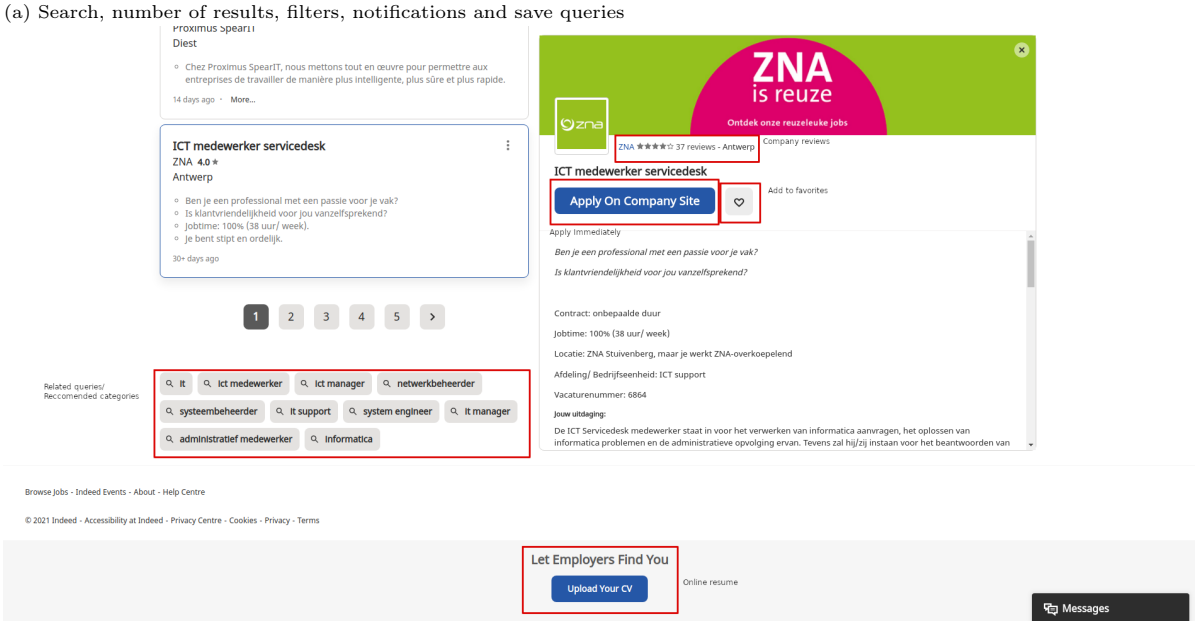
[16] https://github.com/elastic/search-ui

	nationale-vacaturebank	werk.nl	Young-Capital	Werkzoeken	Indeed	Job-bird	Monster-board	LinkedIn
Keywords ⋆	✓	✓	✓	✓	✓	✓	✓	✓
Function ⋆	✓	✓	✓	✓	✓	✓	✓	✓
Location ⋆	✓	✓	✓	✓	✓	✓	✓	✓
Company Name ⋆	✓	✓	✓	✓	✓	✓	✓	✓
Save Queries	✓							
Favorites ⋆	✓		✓		✓			✓
Filter Catagories ⋆	✓	✓	✓		✓	✓	✓	✓
Recommend Catagories ⋆		✓	✓	✓	✓		✓	✓
Related Queries ⋆					✓	✓		✓
Number of results ⋆		✓	✓	✓	✓			
Compare Salary				✓				✓
Similar Vacancies ⋆	✓		✓		✓			✓
Notifications ⋆			✓		✓			✓
Apply Immediately ⋆	✓		✓		✓			✓
Online resume ⋆	✓			✓	✓			✓
CV Search ⋆	✓	✓		✓	✓	✓		✓
Add Vacancy ⋆	✓	✓	✓	✓	✓	✓		✓
Matching			✓					
Company Review ⋆			✓		✓			✓
Sort Candidates					✓			
Message Platform ⋆		✓	✓		✓			✓
'Im ready to work' tag ⋆					✓	✓	✓	✓
Email to potential matches					✓		✓	✓
Social Platform								✓
Save queries ⋆		✓			✓			

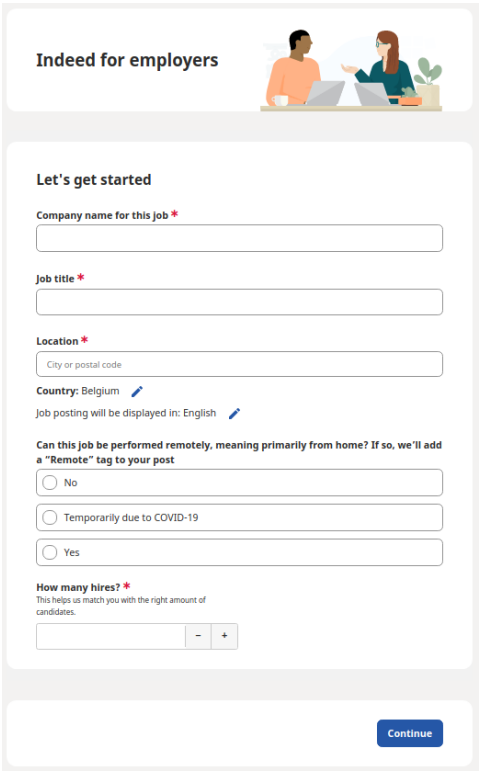
Table 2.9: Features found on job search websites



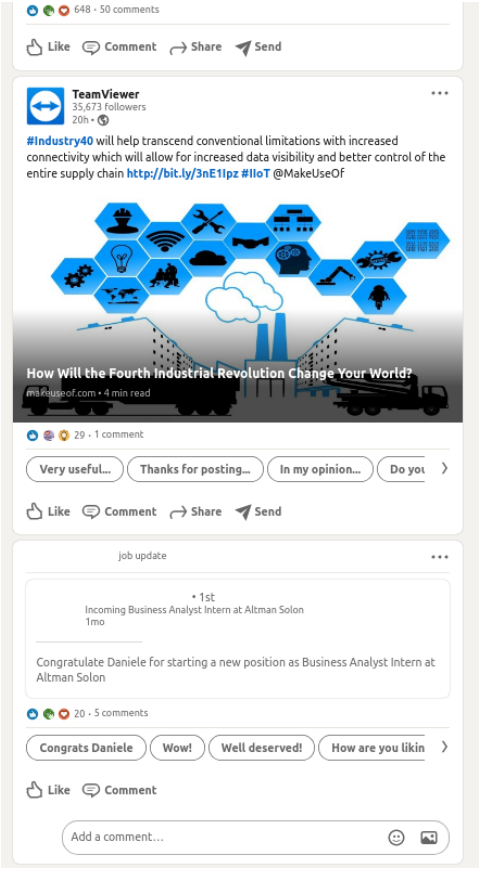
(a) Search, number of results, filters, notifications and save queries



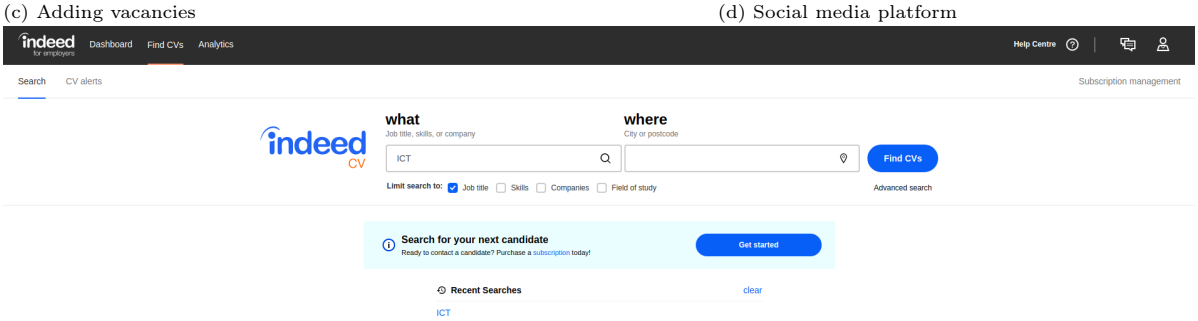
(b) Company reviews, apply immediately, add to favorites, related queries, recommended categories and online CV



(c) Adding vacancies



(d) Social media platform



(e) Searching CV's

Figure 2.3: Features labeled on search page

2.7. Final requirements

With the above investigation in related work in section 2.6 in mind, we have presented a list of requirements to our client and discussed these with them. They responded with some additional requirements, such as the functionality to search and add resumes, which we have added to our final requirements list below. It was decided to rank them according to the MoSCoW model in order to have a clear prioritization and prioritize them by the categories defined below. The final list of requirements can be found in table 2.10.

- Must have
- Should have
- Could have
- Won't have

Importance	Requirement
Vacancies	
	Recruiters can search for vacancies
	Recruiters can filter on categories
	Recruiters have access to the tool via a web interface
	Vacancies must be retrieved from all current associated companies' website
	Vacancies must be retrieved at a regular interval
	Companies are able to upload vacancies straight to the web interface
	Scraped vacancies can be edited through the interface
	Show number of results when filtering
	Recruiters can save favorite vacancies
	Recruiters can see similar vacancies
	Recruiters can save vacancy queries
	Recruiters can compare salaries
	New companies will be automatically add to the scraping process
	Employees will be automatically matched to a vacancy
	Recruiters can upload blog posts
Resumes	
	Allow recruiters to search for resume
	Recruiters can upload an employees resume
	Allow recruiters to send messages about resumes
	Allow recruiters to see which employees are looking for a job
	Allow recruiters to save their queries for resumes
Interface	
	Users can login
	Users cannot register themselves but should be invited
	Users can view the companies associated to Rotteram Werkt
Non-functional requirements	
	The web scraping tool must be tested on all websites of the participating companies
	Code must be easily extensible to support more vacancy websites
	All frameworks and programming resources must be free of charge
	The final product must be able to run on a VPS running Linux
	Use frameworks that have a subscription fee

Table 2.10: Requirements evaluation

2.8. Design goals

Based on the requirements, a number of design goals have been created. These goals will be used at the end of the project to evaluate the quality of the final product.

2.8.1. Security and privacy

Given that the system will contain personal data, it is important that the chance of leaking data is minimized. Therefore, it is important that the data stored cannot be accessed by third-parties. Furthermore, due to GDPR regulations it is important that no personal data will be logged in order to make sure that the 'right to be forgotten' can be enforced.

2.8.2. Maintainability

The system should be easy to maintain. Given the volatility of the scrapers due to the possibility of companies changing their website significantly, having to update a scraper should require minimal effort. Furthermore, as *Rotterdam Werkt!* may expand in the future, it should also be possible to easily add new scrapers to the system. The client also suggested that the system might be maintained by future students. Therefore it is crucial that the code is documented properly and that it is maintainable.

2.8.3. Ease of deployment

Since the client mentioned they will most likely hire students to maintain this system in the future. It should be reasonably easy for them to deploy the system. This means that the deployment method cannot be overly complex and must be clearly described in the documentation.

2.9. Approach

In order to implement the requirements listed in section 2.7, decisions needed to be made on the specific approaches to be taken. This section will discuss the approaches that were taken in sections 2.9.1 to 2.9.4 together with some of the risks involved in section 2.9.5. Finally, the section will end with a preliminary planning for the implementation and reporting stages in section 2.9.6.

2.9.1. Development methodology

The usage of the correct development methodology will help us to divide the project into smaller pieces as well as being able to finish within the scheduled time.

The first methodology worth considering is Scrum [23]. This is an agile methodology, which means that a working version of the project is reviewed early and often. The requirements could thus be changed throughout the project, every time a working version is delivered. This allows the product owner to finely tailor the final product to their needs, even in later stages of the software project. Scrum works in iterations of a specified length, usually weekly or bi-weekly, called sprints. At the start of a sprint, the team will come together and divide the tasks that need to be completed during the sprint. At the end of the sprint, a working version is delivered and evaluated. Any necessary changes will be noted for the next sprint such that the team can adapt and continue working on the project [24].

A second methodology is the Waterfall approach [23]. This is a relatively simple approach where all requirements are defined at the beginning of the project. After this, the project runs in predefined phases which need to be well documented in order for the approach to work. The Waterfall approach is rather rigid as it is costly, both with regards to resources as well as time, to alter the requirements. More concretely, if a requirement is to be changed in a given phase, all previous stages have to be executed again with the altered requirement in order to get the desired outcome [25].

For our project the Scrum approach will be taken. This approach allows us to create a final product which is tailored very closely to the clients' wishes, compared to the waterfall approach which causes a great technical deficit when requirements are changed. Furthermore, the iterations allows us to better split up the tasks gradually as we progress through the project in order to prioritize certain tasks.

2.9.2. Documentation

The code will have method- and class-based documentation. This documentation will make it easier for future developers to better understand the code and thus make it easier to maintain the platform in the future. Furthermore, a deploy guide will be added in the `README.md` file of both the website and

the scraper. This guide will contain a detailed explanation of how this project should be deployed. This should make it easier for future maintainers of the project to integrate new features.

2.9.3. Version control

As a version control system we have the choice between GitLab[17] and GitHub[18], since these are the ones we have the most experience with and we have limited resources, thus payed options are not viable. We have decided to use GitLab as this gives us access to more pull-based development features such as running unlimited pipelines for free. We will be using pull-based development, this means that for every task during the week, an issue will be created. The developer will then 'branch off' from the main branch in order to complete the new task. Once the task is finished, the developer will submit a merge request. At least one other member of the team will inspect the merge request and judge it on code quality, style and structure, as well as checking the documentation for clarity. Within the merge request, all tests and code quality tools will be run by the CI pipeline to ensure new features do not break existing ones. Once everything passes and the reviewer is satisfied, the new code can be merged into the main branch. This ensures that we always have a fully working version of the final product and that new code adheres to the standards set by us.

2.9.4. Static code analysis

As a static code analysis tool for the python code, Flake8[19] will be used. Given that the tool is designed specifically for Python and it is still being maintained, this will give a reasonable idea of the code quality. Flake8 also does linting and will therefore also make sure that the code is formatted correctly [26]. Furthermore, Flake8 can also be run in our GitLab pipelines and will fail the pipeline if the code is of insufficient standard.

For React, the standard code analysis tool provided by 'react-create-app', ESLint, is used. ESLint analyzes the code in order to find problems and it is fully customizable [27]. It was decided to use the default configuration as this sufficiently met our needs.

2.9.5. Risk analysis

The usage of the final system may present certain risks. In this section these risks will be assessed and discussed.

Changes in the company websites

Each scraper for each website will be designed for the current layout of the website. This means that if a website changes drastically in terms of layout, URL and/or architecture, either during the project, or after the project, the scraper will not work any more. If this happens during the project this problem can be solved by us, even though this will create a delay in the planning. When this happens after the project, the client will need to hire new software engineers in order to fix the scraper such that it can scrape the specific website again.

Addition of new companies

When a new company joins *Rotterdam Werkt!*, the vacancy website for this company will need to be added to the scraper. This means that a new spider will need to be created for this company. In order to do this, a new software engineer will need to be hired in order to do this. This might result in the project being discontinued as it will need a relatively high level of maintenance.

2.9.6. Planning

In order to be able to plan the stages of this project, we have created a preliminary schedule in table 2.11. We have added all known deadlines for the research report, SIG evaluation, thesis report and final presentation. Furthermore, we have also divided the implementation phase into our 3 main components and added deadlines for each component. This means that we can dedicate the period after Christmas to connecting these components and fixing any last bugs that may arise from this.

[17] https://gitlab.ewi.tudelft.nl
[18] https://github.com
[19] https://flake8.pycqa.org/en/latest/

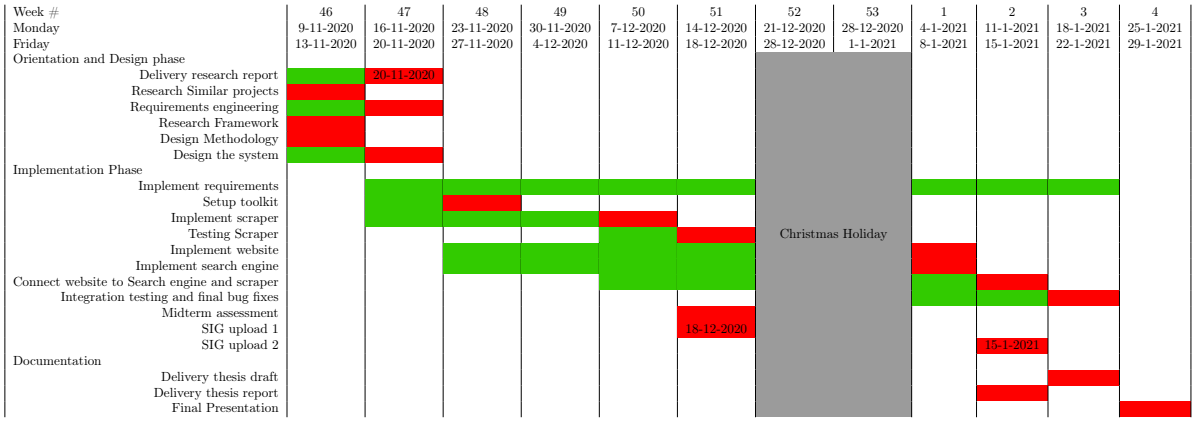


Table 2.11: Preliminary schedule

3 Design

3.1. Overview

In this chapter the design of the software will be considered. The chapter will start with a discussion on the general architecture of the product in section 3.2. Afterwards, each component will be discussed individually in sections 3.3 to 3.6.

3.2. Architecture

This section discusses the architecture design of the system. Figure 3.1 shows the final architecture. Users can interact with the application through the user interface. This is connected to the back-end via a reverse proxy. This proxy will redirect requests made to the back-end and forward them to either the REST-server or Elasticsearch. The REST-server will store received vacancies from either the user interface or the scraper in Elasticsearch and the database. Lastly, the REST-server and Elasticsearch will produce logs, which will be processed by Logstash and stored back in Elasticsearch. Then, Kibana is able to access these through their respective indexes and analyze them.

Given all the different services used within the application, it could be considered as a microservice architecture [28]. This has the advantage that each service can run autonomously. Another advantage is robustness. That is, if one service were to be taken out or fail, the other services would still continue to function. Because each service functions independently, they can be switched out at any point in time for a similar compatible service without needing to redeploy or redesign the whole stack. Furthermore, it is also easier to build and maintain a microservice architecture. Each service will have a smaller code base, especially compared to a monolithic approach where there is one big code base, meaning that maintainability will increase [28].

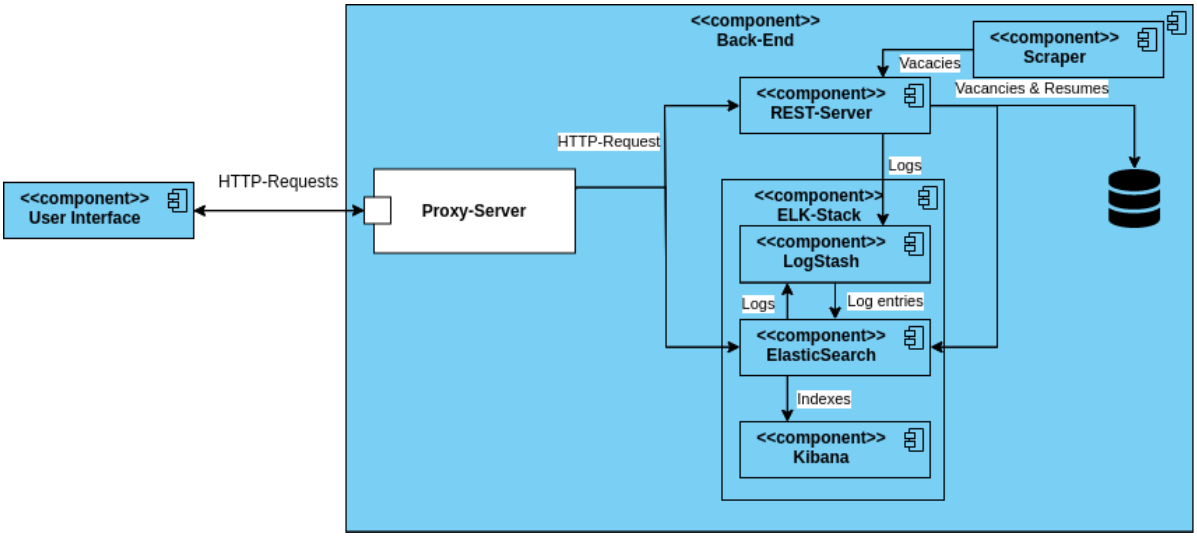


Figure 3.1: Overall component diagram

3.3. Scraper

The scraper will crawl the vacancy web pages of the *Rotterdam Werkt!* organizations and extract relevant data. For each organization, it includes a spider that will scrape a specific website.

It starts at the vacancy overview page and will collect the URLs of the vacancy detail pages. These detail pages will then be scraped for relevant information, such as the title, hours, contact information and department. A complete list of collected data fields can be found in table 3.1. If necessary, data is sanitized. Finally, all extracted information is sent to the REST-server, described in section 3.5.

Field name	Description
COMPANY_TITLE	Identifier of the scraped company
CONTACT_EMAIL	Email address of recruiter associated with vacancy
CONTACT_NAME	Name of recruiter associated with vacancy
CONTACT_PHONE	Phone number of recruiter associated with vacancy
DEPARTMENT	Company department to which vacancy belongs
EDUCATION_LEVEL	Education level required to apply
END_DATE	Vacancy end date
FULL_TEXT	All plain text in vacancy without formatting and without HTML tags
HOURS_PER_WEEK_MAXIMUM	Max. number of work hours per week according to vacancy
HOURS_PER_WEEK_MINIMUM	Min. number of work hours per week according to vacancy
IS_SCRAPED	True if vacancy was scraped, False if recruiter uploaded vacancy
JOB_LEVEL	Job level (e.g. internship, junior, medior, senior, ...)
JOB_LOCATION	Main physical location of job
JOB_TYPE	Type of employment (part time, full time, ...)
SALARY_CURRENCY	Currency the salary is given in (e.g. euros, dollars, ...)
SALARY_MAXIMUM	Upper bound on salary provided by job
SALARY_MINIMUM	Lower bound on salary provided by job
SPIDER_NAME	Identifier of the spider that has been run
TITLE	Vacancy title
URL	Vacancy URL

Table 3.1: List of fields extracted from the vacancy web pages by the scraper

3.4. Database design

The final database schema is depicted in figure 3.2. Given that we are using objects in the REST-server, the final database schema is created for an object-relational database. The database schema mainly revolves around three object types: vacancy, resume, and recruiter. Each of these object types will be discussed in more depth below.

Vacancy

The vacancy object type contains all the information which can be scraped from the websites. Vacancy has a one-to-many relationship with company. This allows us to keep track of the companies, and by extend the recruiters, who are associated with a vacancy. Furthermore, vacancies have a many-to-many relationship with certificates. This allows for easy retrieval of all the vacancies which are associated with a specific certificate and thus allows Elasticsearch to easily filter on them. Moreover, this also allows users to add already existing certificates to their vacancies without needing to duplicate them. In future work, this relationship could also be applied to other elements in order to make them filterable.

Resume

The resume object contains the information needed to form a full resume. The model is inspired by the LinkedIn resume functionality. The parent object contains the basic personal information. Resume also has child objects in the form of a one-to-many relationship with education, skills and experiences. This means that a user can have several of these objects in their resume without having to duplicate the resume itself.

Recruiter

A recruiter can also be seen as a user in this system. The recruiter is linked to a company. This authorizes the recruiter to edit only the vacancies which are linked to this company. Furthermore, a recruiter is also associated to a Password Request object, which allows the recruiter to change their password. The recruiter will receive a unique id from this Password Request which they can use to change their password.

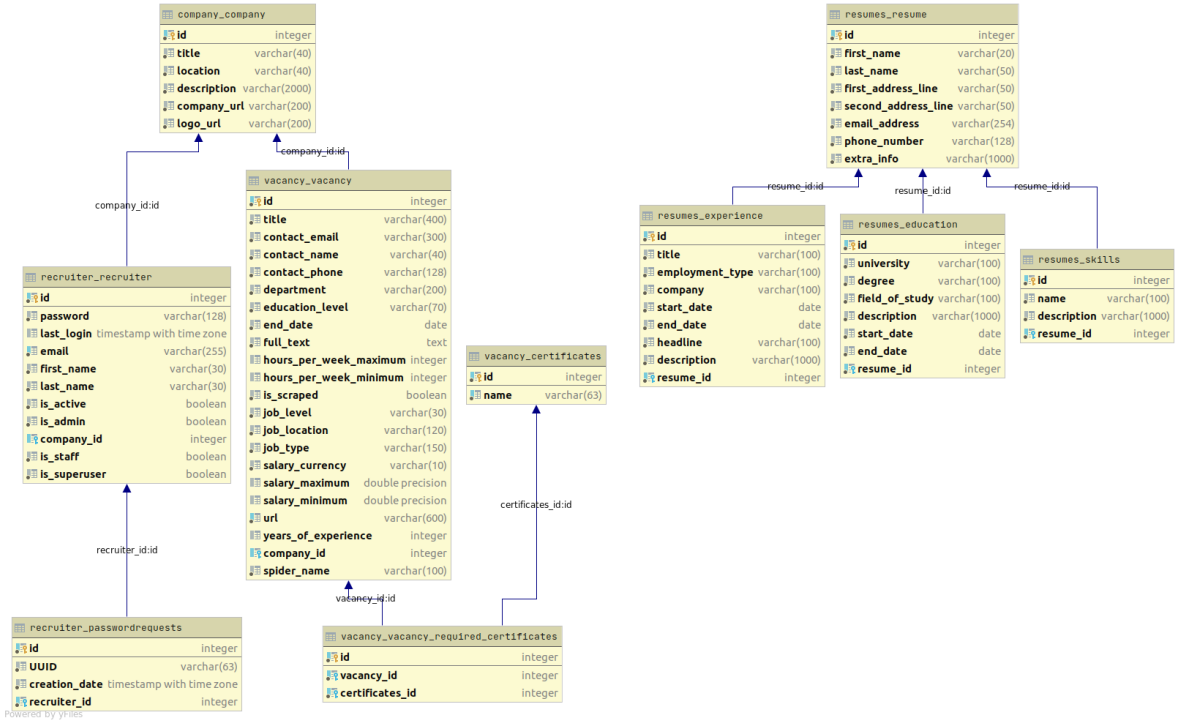


Figure 3.2: Database diagram

3.5. REST-server

The REST-API provides a set of requests that can be made by the client to the REST-server. It was decided to use REST as standard for the API since this is a popular standardized method for communication between web services [29]. The API has functionality to GET, POST, PATCH and DELETE objects via requests. Below is a list of API calls which should be implemented.

VacancyList

Requests	Parameter	Returns
GET	-	List of all vacancies
POST	-	-

VacancyDetail

Requests	Parameter	Returns
GET	Primary Key	More detailed information about vacancy
PATCH	Primary Key	-
DELETE	Primary Key	-

CompanyList

Requests	Parameter	Returns
GET	-	List of all companies
POST	-	Adds a company

CompanyDetail

Requests	Parameter	Returns
GET	Primary Key	More detailed information about Company
PATCH	Primary Key	Edits a company
DELETE	Primary Key	Deletes a company

ResumeList

Requests	Parameter	Returns
GET	-	Returns a list of all resumes
POST	-	Adds a resume

ResumeDetail

Requests	Parameter	Returns
GET	Primary Key	More detailed information about Resume
PATCH	Primary Key	Edits a resume
DELETE	Primary Key	Deletes a resume

ResumeEducationListView

Requests	Parameter	Returns
GET	Resume Primary Key	Returns all educations from a resume
POST	Resume Primary Key	Adds an education to a resume

ResumeSkillsListView

Requests	Parameter	Returns
GET	Resume Primary Key	Returns all skills from a resume
POST	Resume Primary Key	Adds a skill to a resume

ResumeExperienceListView

Requests	Parameter	Returns
GET	Resume Primary Key	Returns all experiences from a resume
POST	Resume Primary Key	Adds an experience to a resume

3.6. Front-end design

In this section, we will explain the design of the front-end. The front-end will have two distinct purposes. Firstly, it should allow users to interact with the vacancies and resumes. In other words, create, edit and delete vacancies and resumes. Secondly, it should allow users to search for vacancies and resumes.

3.6.1. Interaction with resumes and vacancies

For users to be able to interact with resumes and vacancies, screens are needed in order to display forms. These forms allow the user to create new resumes and vacancies, as well as edit already existing ones. Since a resume object can have several child objects for education, experience and skills, separate forms will be created for these objects such that each child object can be updated independently without causing all child objects to be updated.

3.6.2. Searching for resumes and vacancies

Given that users should be able to search for resumes and vacancies, there are a number of elements available to them. This includes a search field, facets (also known as filters), specifying the number of elements to be displayed and a paging element in case a search returns a large result set. Furthermore, the users will be able see the basic initial data displayed on the search page itself, but will also have the option to view the whole resume or vacancy by clicking on the search result.

4

Implementation

4.1. Overview

This chapter will discuss the implementation details of each component in the system. Section 4.2 discusses the implementation of the scraper. In section 4.3 the implementation of the back-end will be discussed together with the data and authentication flow, followed by the implementation of the front-end in section 4.4. Afterwards, section 4.5 will describe the logging which was implemented and its purpose. Section 4.6 will describe how all the components will be connected together in production through containerization. Lastly, section 4.7 will discuss the testing approach taken for this project.

4.2. Scraper implementation

The scraper has been written in Python using a web-crawling framework called Scrapy. Its architecture is based on self-contained 'spiders', which can crawl websites for data. For each member of the *Rotterdam Werkt!* organization, a spider was written to crawl the web page(s) on which their vacancies are listed. If their content was loaded dynamically, an additional framework, Selenium, was also used.

4.2.1. Static pages

For each spider, Scrapy will make initial requests to the URLs provided in the `start_urls` list and call the `parse` method when the response is returned by the server. The `start_urls` list most often consists of a single URL pointing to the overview page that lists the first x vacancies for a given organization. In the `parse` method, hyperlinks to the detail pages of listed vacancies are extracted from the HTML body using Scrapy's built-in XPath and CSS selectors. These links are then added to Scrapy's request queue with a callback to a `parse_vacancy` method that will extract the relevant vacancy data. Additionally, for organizations with their vacancies listed on multiple pages, the link to the next page is extracted and added to the queue. The `parse_vacancy` method typically also uses XPath and CSS selectors to extract vacancy data such as the title, department, salary, hours, required education level and contact information.

4.2.2. Dynamic pages

Several organizations did not have their vacancies contained within the original HTML page, but rendered later using JavaScript, which Scrapy cannot extract out of the box. In these cases, Selenium was used as a time-saving measure, as programmatically recreating the requests made by JavaScript and extracting the relevant data within may prove rather time-consuming. This decision was motivated by the Scrapy documentation which reads: "You can reproduce any request with Scrapy. However, some times reproducing all necessary requests may not seem efficient in developer time" [30].

Some websites had their vacancy data enclosed within the JavaScript itself, such as the example in section 2.3.2, and others had them loaded in dynamically using AJAX and a JSON API. Where possible and `robots.txt` 'allowed' us, data was obtained from a JSON API directly. For all other dynamic websites, a headless Firefox browser was used, controlled by Selenium WebDriver. By using an actual browser combined with Selenium, the content of the vacancy pages is rendered exactly as

intended by the web designers, while making the data accessible with XPath and CSS selectors. Using Selenium now also has the added benefit of making it easy to add another dynamic vacancy page as it can just be treated as if it was a static page.

4.2.3. Spider automation

The Scrapy framework provides a command-line interface to run and debug spiders. With `scrapy crawl spider_name -O output.json`, the named spider will be executed and the scraped data will be exported to a file in JSON format. This method was used to test individual spiders. Whenever a spider is run, all scraped items retrieved by the spider are automatically passed utilizing an HTTP POST request to the REST-server discussed in section 4.3 using a Scrapy item pipeline[1].

When deployed with the rest of the system, all spiders can be run automatically by calling a single Python file: `rotterdam_werkt_scraper.py`. This method was chosen in order to enable easy scheduling with tools such as cron or Windows Task Scheduler. By default, this Python script will run all spiders sequentially, but it is also possible to run them in parallel by adding them to the `parallel_spiders` list. Initially, it was intended for all spiders to run in parallel, but unfortunately, timeout issues and various other bugs made this unworkable. Therefore, the decision was made to run all spiders sequentially by default to avoid any issues.

4.3. REST-server implementation

The REST-server has been implemented using Django, a server framework written in Python. Django also provides the database we will be using. This REST-server exposes an API for external services to connect to in order to perform REST operations on the data. The API works together with the database using serializers. They are responsible for converting the objects in the database to an object which can be send using an HTTP request. In this case, JSON is used to transfer the data between the client and the server. When a new or updated object is received by the REST-server, it will commit this to the database, as well as update the Elasticsearch index with this object. A high-level UML can be seen in 4.1.

4.3.1. API and serializers

For the API we have used the “Django Rest Framework” library, which provided us with REST API endpoints and object serialization functionality. For every database model the API can access, a serializer must be provided. The serializer provides a way to check the validity of the JSON object sent to the REST API endpoint. The API provides two options for data retrieval, either a single object or a list of all the objects. Therefore, two types of endpoints have been created for the models: the `DetailView` and the `ListView`. Detailviews takes one argument: the entry ID. This ID can be used to select a certain entry and delete it with a `DELETE` request or update it with a `PATCH` request, given the user has the appropriate permissions. ListViews are used to retrieve a list of multiple entries of the database and to add a new entry.

4.3.2. Data storage

As a database option we considered PostgreSQL and MySQL given that we have the most experience with these two. Ultimately, it was chosen to use PostgreSQL[2] as a database. The choice for Postgres compared to MySQL[3] is driven by the fact that Postrgres is an object-relational database, whilst MySQL is a purely relational database. PostgreSQL is therefore more suited to work with an object oriented database schema. Furthermore, Postgres is also better at handling concurrency which is an advantage when the scraper is running and posting several vacancies in a short time span [31], [32].

The Postgres database will store all the persistent data such as users, vacancies and resumes. This allows for data redundancy and data consistency as well as easy retrieval for specific objects by an identifier.

The REST-server will also index the object in Elasticsearch, after committing to the database, such that users are able to query for it. The mappings for vacancies and resumes can be found in appendices E.1 and E.2 respectively.

[1] https://docs.scrapy.org/en/latest/topics/item-pipeline.html
[2] https://www.postgresql.org/
[3] https://www.mysql.com/

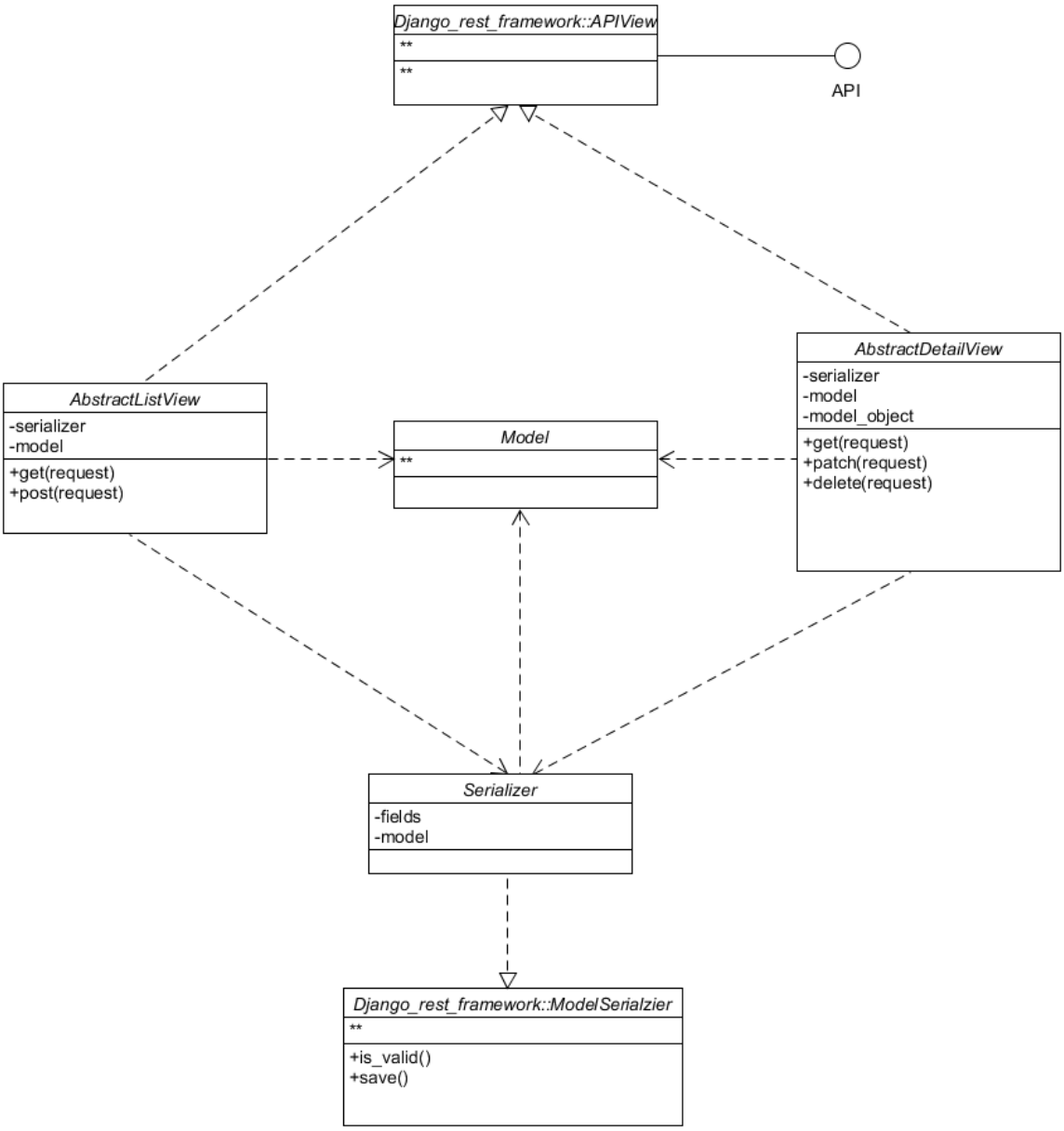


Figure 4.1: High-level UML of REST-server

4.4. Front-end implementation

The two main parts of implementing the front-end are creating forms for users to add and update the data, and creating the search interface to allow users to search for vacancies and resumes. In section 4.4.1 the approach to creating forms will be discussed. Afterwards, in section 4.4.2, the search implementation will be discussed.

4.4.1. Form implementation

In order to create forms with validation, a React library called Formik was used. Formik provides a state for forms. In other words, it keeps track of the values, errors and visited fields in a form [33]. Furthermore, it can also validate the input using a validation schema before the input is passed on to a submit method.

For the validation schema Yup was used. Yup is a library which provides a schema builder for value parsing and validation [34]. It allows Formik to do form validation before submitting. This ensures a better defensive programming style to be adopted since no wrongly formatted data can be sent to the REST-server from the front-end.

In the submit method defined in Formik object, the interaction with the API is performed. It will create a POST request to the API using the Axios[4] library. This request returns a Promise object which is resolved during the submit. If the API responds with `200 OK`, the user will be redirected to the correct following page, usually to view their added or edited data. The server may also return a `401 Unauthorized` error. In this case, the user does not have permission to add or edit the data, meaning they will be redirected to an error page. In the unlikely case that the object sent to the API is not valid, the server will respond with `400 Bad Request`. In this case, a general error message will be displayed at the bottom of the page to notify the user of this problem.

4.4.2. Search implementation

For the search implementation, the Search-UI library was used. This is a library provided and maintained by Elasticsearch. It provides a way to keep track of the state of the search functionality, as well as some basic components such as facets, results page and paging. However, due to the complex nature of some of the objects we are using, such as nested objects, the built-in query building functionality provided by Search-UI and Enterprise Search[5] could not be used as this does not support nested objects. Luckily, the library provides a way of overriding this. This means that full control can be taken over the creation of the search query performed by the search engine. A number of example queries are listed in appendix G.

4.5. Logging

In order to facilitate the future development of this project, logging has been added. With logging, it is possible to do bug tracking, such that frequent errors can be detected. Furthermore, data can also be collected in order to further optimize the search engine. Therefore, the performed queries can be logged to create aggregated statistics on the frequency of certain queries. In section 4.5.1 the approach to error logging is described, afterwards in section 4.5.2 the query logging approach is illustrated.

4.5.1. Back-end logging

In order to be able to identify certain recurring errors in the application, logging has been added to the back-end in case the API returns an error. To efficiently do logging, the `logging` package, provided by Django, has been used. This package allows for several levels of logs: 'info', 'debug', 'warn' and 'error'. Whenever the API returns a `4xx error` to the client, it will log the errors generated by the serializer or API.

These logs are converted to the following format: `$date$ $level$-$message$`. They are then stored in a file accessible to Logstash[6]. These will be read by Logstash and parsed into documents that will be stored in Elasticsearch. This allows Kibana to query Elasticsearch for these logs. Afterwards, the administrator will be able to group and evaluate these logs in order to identify often recurring errors and which in that case may indicate a bug.

[4] https://github.com/axios/axios
[5] https://www.elastic.co/enterprise-search
[6] https://www.elastic.co/logstash

Figure 4.2 shows a use-case example of displaying logs in Kibana. The pie-chart shows the log levels detected in the logs in the inner level, and the error tag per log level on the outside. This pie-chart was generated after the first time the scraper was run in combination with the back-end. Thanks to this pie-chart, it was possible to identify a problem and also gave some indicators to where this might lie.

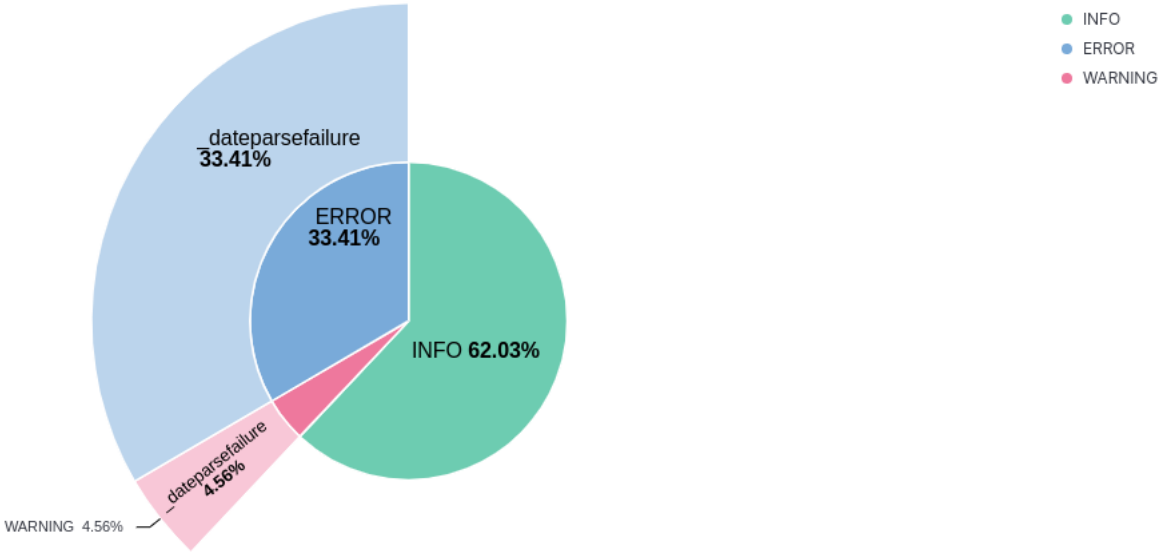


Figure 4.2: Kibana display error

4.5.2. Query logging

Due to certain limitations described in section 5.4 regarding our search engine evaluation, we strongly recommend that this is further investigated in the future. In order to do this efficiently, it is important to have an idea of the queries which are executed in the search engine. It was therefore decided to log all the queries which are performed.

These logs, unlike the back-end logs, are more complex in format with the pattern `[&date&][&loglevel&][&-25c1.&] [&node_name&] &m&`, where `&m&` refers to the query information with the total amount of time taken for running the query, which fields were queried and what the query was. Being able to access this information gives the administrator of the application the possibility to collect more data for future search engine evaluations as well as identify potential problems with the system. For instance in case of sudden spikes in query execution time.

Figure 4.3 shows an example of the statistics created with these query logs in Kibana. Each bar represents a query and the length of the bar represents the number of times this query is performed. When hovering over the the bars, the full query will become visible. See appendix G for the actual queries related to this chart.

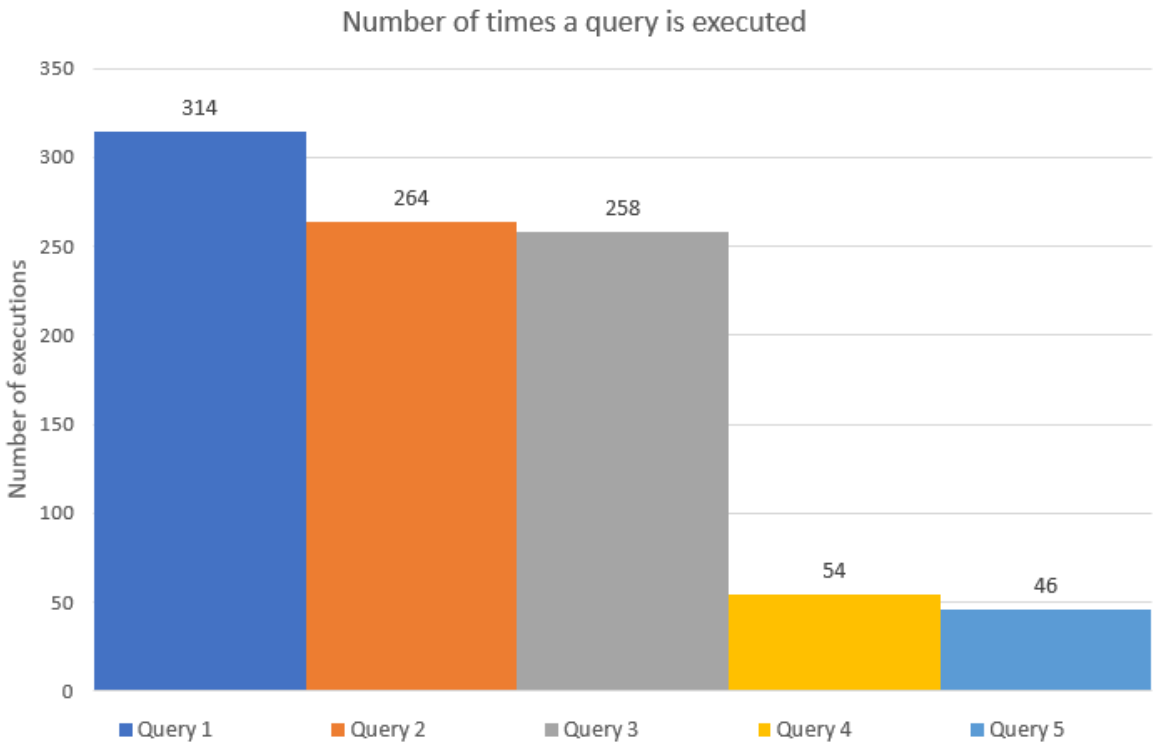


Figure 4.3: Query logging statistics, see appendix G for query details

4.6. Docker implementation

In order to provide a way to do manual integration testing as well as rapid and easy deployment, we decided to containerize our application using Docker[7]. We have created a container for each service that we have implemented: REST-server, front-end and scraper. We have used the dockers provided by Elasticsearch in order to gain access to Elasticsearch, Kibana and Logstash without needing to install them locally. These have been configured using the `docker-compose`[8] functionality provided by Docker. This provides the developers with easy access to the code base in order to implement new features without needing to go through an extensive setup processes. Below, we will describe the images we have used in order to run our different services.

4.6.1. REST-server

The REST-server image is created through the `Dockerfile.backend`. `python3.8.3-alpine` is used as a base image, this provides us with an instance of Python version 3.8.3 installed in Linux Alpine. Alpine was used as this provides us with a security oriented and lightweight OS and thus also a secure and lightweight container [35]. This, however, meant that we needed to install `gcc`, `postgresql-dev`, `python3-dev` and `musl-dev` ourselves as these are necessary dependencies to run our code which are not provided by `pip`.

4.6.2. Front-end

The front-end image is created through a two stage docker file described in `Dockerfile.frontend`. During the build stage we use the `node:13.12.0-alpine` image. This provides the necessary node dependencies in order to build the React project. Once the build is complete, these files are copied to a `nginx:stable-alpine` image. This allows us to use NginX[9] in order to serve our previously generated files to a client. In the NginX config we have specified from where to serve our React application as well as the needed proxies to the back-end and Elasticsearch. This means that we

[7] https://www.docker.com/
[8] https://docs.docker.com/compose/
[9] https://www.nginx.com/

have full control over the exposure of our Docker's endpoints and we can thus decide which endpoints should or should not be exposed.

4.6.3. Scraper

The scraper image is, similar to the back-end, based on the `python3.8.3-alpine` image. During the creation of the image, we schedule the Python script to be run every day at 2:30 AM using Crontab[10]. A command can be given to this container which will be run once, this will allow the maintainers to make an initial run of the scraper to populate the database if needed.

4.6.4. Security

Other than ease of deployment, the use of Docker also provides a certain level of security. This is considered very important due to the personal nature of the data which is being stored. In literature it is argued that Docker containers are fairly secure at their default configuration, nonetheless care should be taken through the use of only trusted images and limit their operating level to 'non-privileged'. Furthermore, tools such as AppArmor or SELinux can be used as hardening solutions in order to further secure the application [36]. The usage of these hardening solutions for the Linux kernel is something which should be investigated further, but fell outside the scope of this project.

4.7. Testing

Knowing that the scraper could become a fragile tool for the application, it became clear that we needed an extensive testing method in order to make sure the system was as failure-proof as possible. Section 4.7.1 discusses the approach taken in order to achieve this. Furthermore, in order to achieve correct functionality in the back-end and the front-end, a testing framework was also setup for these. These frameworks are discussed in section 4.7.2 and 4.7.3 respectively. The search engine will not be tested in a way similar to the scraper, the front-end and the back-end, but will be evaluated in section 5.4. To be able to distinguish whether our software has been sufficiently tested, and thus meet the requirement regarding testing in table 2.10, the quality of our test suites should be gauged. This has been done using a coverage tool called Jest[11] for the REST-API and Coverage.py[12] for the scraper.

4.7.1. Scraper testing

The tests for the scraper predominantly comprise unit tests, which can be subdivided into two main categories: auto-generated unit tests and manually written unit tests. The former has been used for the parse methods in each spider that heavily relies on the Scrapy framework, whereas the latter has been used for testing helper methods and utility functions and classes. Each category is discussed in their respective subsection. Besides unit tests, manual testing has also been applied to situations in which unit tests could not suffice.

Auto-generated unit tests

One of the reasons for choosing Scrapy over other alternatives is a package called `scrapy-autounit`[13] for automated testing, which proves to be especially useful for most parse methods in each spider. While enabled, test fixtures and test cases are automatically generated when a spider is run. In other words, the moment the spider is run while automated test generation is enabled, Autounit captures the output of the spider and the responses to the requests the spider makes. Then, when the tests are run, the output of the spider at that time is compared to the saved output, using the emulated internet connection.

However, Autounit is not a cure-all method of test generation. For instance, some spiders require the use of a headless browser to support vacancy websites that use JavaScript to dynamically load their content. Even though Scrapy is compatible with this, Autounit unfortunately is not. To solve the issue, parse methods that rely on such a browser have been tested by manually inspecting the output JSON data. Moreover, it must be noted that these are regression tests and do not account for changes in the web pages or URLs. They only test if changes made to the spider changes its output.

[10] https://crontab.guru/
[11] https://jestjs.io/
[12] https://coverage.readthedocs.io/en/coverage-5.3.1/
[13] https://github.com/scrapinghub/scrapy-autounit

Unit tests
Besides using auto-generated tests, unit tests have also been written manually to cover the spiders' helper methods as well as the utility classes and functions. Most tests have been parameterized, as parameterized tests reduce code duplication and improve readability.

4.7.2. REST-server testing
To test the REST-server, we have decided to use the Django REST Framework API's standard test classes. The Django model also provided us with a `request_factory` which made simulating requests simple. Using these tools we have tested every API endpoint.

4.7.3. Front-end testing
For the testing of the front-end, it was decided to use the React Testing Library. This testing library comes installed automatically through the 'react-create-app' functionality provided by React. It allows for the creation of unit tests for each component and hook of the application. The React Testing Library allows these components to be rendered and interact with them during a unit test. In order to verify the correct interaction, Jest[14] can be used. Jest allows for assertions on the state of components to be made using their Matcher system, as well as mock certain methods in order to verify behaviour, such as making requests to the API.

[14] https://jestjs.io/

5

Product evaluation

5.1. Overview

In this chapter, the final product will be evaluated. Section 5.2 provides an evaluation of the requirements. For each requirement, it will be decided whether it has been met or not. Afterwards, also the design goals will be evaluated in section 5.3 in order to make sure we have met them. In section 5.4 the evaluation of the search engine will take place. Section 5.5 will discuss the evaluation of our software quality by the Software Improvement Group (SIG). Lastly, ethical implications will be discussed in section 5.6.

5.2. Product evaluation

The final product will be evaluated based on the requirements created at the start of the project, see section 2.7. An evaluation of these requirements can be found in table 5.1. From the evaluation it becomes clear that all must-have and some Should-have requirements have been completed. This means that the final product can be considered as a 'minimal-viable-product' which still leaves improvements to be made in the future. Screenshots of the final product can be seen in figure 5.1

5.3. Design goal evaluation

In this section the design goals that have been set in section 2.8 will be evaluated to make sure they have been met.

5.3.1. Security and privacy

In order to reduce the chance of data leaking and unwanted access to the services, Docker images are used given that literature suggests that they are fairly secure [36]. Furthermore, extra caution has been taken not to log any personal data when query and error logging is done. This ensures that 'the right to be forgotten' can be enforced. We therefore believe that we have met this design goal. Nonetheless, security considerations are not only important during the design and implementation phase, but should also be taken into consideration during deployment. For example, during deployment extra care should be taken that the website is only accessible through `HTTPS` and and users should be made aware that they are responsible for using sufficiently strong passwords for their accounts.

5.3.2. Maintainability

We believe that the system is easy to maintain, as the scrapers have been designed in a way that new scrapers can be added with minimal effort. Maintainers will need basic Python skills and the maintenance process requires time, which may entail monetary investments.

Furthermore, extra effort has been put in to make sure that there is sufficient documentation and the code is comprehensible. This is reflected in the favorable maintainability score we received from the Software Improvement Group (SIG), as discussed in section 5.5.1. Furthermore, SIG's suggestions have also been processed in order to further improve the maintainability and thus make sure this design goal is met.

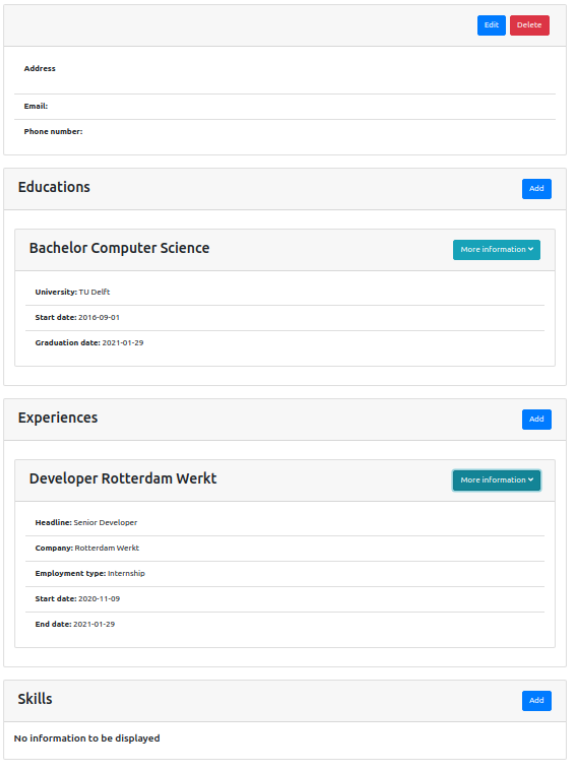
	Requirement	Done/ Not done	Comment
Vacancies			
	Recruiters can search for vacancies	Done	
	Recruiters can filter on categories	Done	Recruiters can filter based on: salaries, certificates and companies
	Recruiters have access to the tool via a web interface	Done	
	Vacancies must be retrieved at a regular interval	Done	
	Vacancies must be retrieved from all current associated companies' website	Done	The websites of KOTUG and Ahoy are not considered here since they had no vacancies on their websites and therefore could not be scraped.
	Companies are able to upload vacancies straight to the web interface	Done	Can be done by recruiters associated with the company.
	Scraped vacancies can be edited through the interface	Done	This is done, however when the vacancy is scraped it might override the changes.
	Show number of results when filtering	Done	
	Recruiters can save favorite vacancies	Not done	Insufficient time to finish requirement.
	Recruiters can see similar vacancies	Not done	Insufficient time to finish requirement.
Resumes			
	Allow recruiters to search for resume	Done	
	Recruiters can upload an employees resume	Done	Recruiters can create resumes for their employees. Here they can add basic information, educations, previous experience and skills.
	Allow recruiters to send messages about resumes	Not done	
	Allow recruiters to see which employees are looking for a job	Not done	Insufficient time to finish requirement.
	Allow recruiters to save their queries for resumes	Not done	Insufficient time to finish requirement.
Interface			
	Users can login	Done	
	Users cannot register themselves but should be invited	Done	New users should be added by an admin account, they will then receive an email to complete their account and create a password.
	Users can view the companies associated to Rotteram Werkt	Done	The admin can add companies to the platform with basic information.
Non-functional requirements			
	The web scraping tool must be tested on all websites of the participating companies	Done	The websites of KOTUG and Ahoy are not considered here since they had no vacancies on their websites and therefore could not be tested.
	Code must be easily extensible to support more vacancy websites	Done	New spiders can easily be added and they will then automatically be run.
	All frameworks and programming resources must be free of charge	Done	
	The final product must be able to run on a VPS running Linux	Done	The final product is able to run in Docker which will run on most popular Linux distributions.

Table 5.1: Requirements evaluation

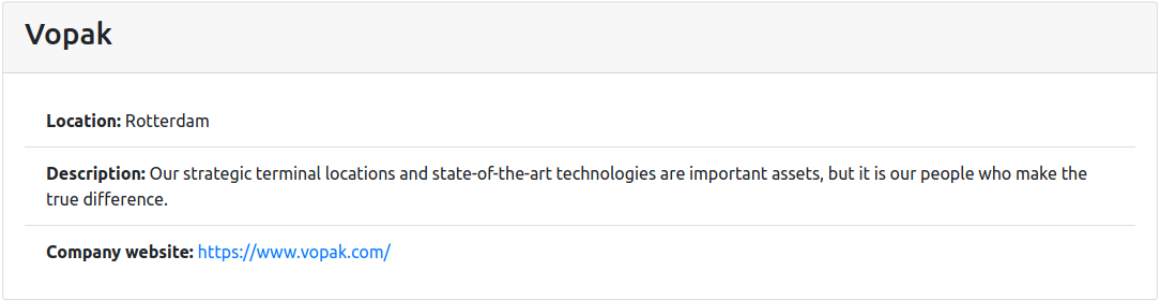
5.3.3. Ease of deployment

Given that the client mentioned that they will hire students to maintain the system in the future, a simple and easy way of deploying the application is needed. In order to meet this goal two actions have been taken. First of all, the deployment is done using `docker-compose`. This gives the developer the option to, with a single command, restart a single service or the whole stack, when components of the software have been update. Secondly, there is an extensive deployment guide in the `README.md` file which explains in details how the system should be configured. Certain team members did not have any Docker experience when starting the project, and when they followed this guide, they were able to setup and run the project without any trouble.

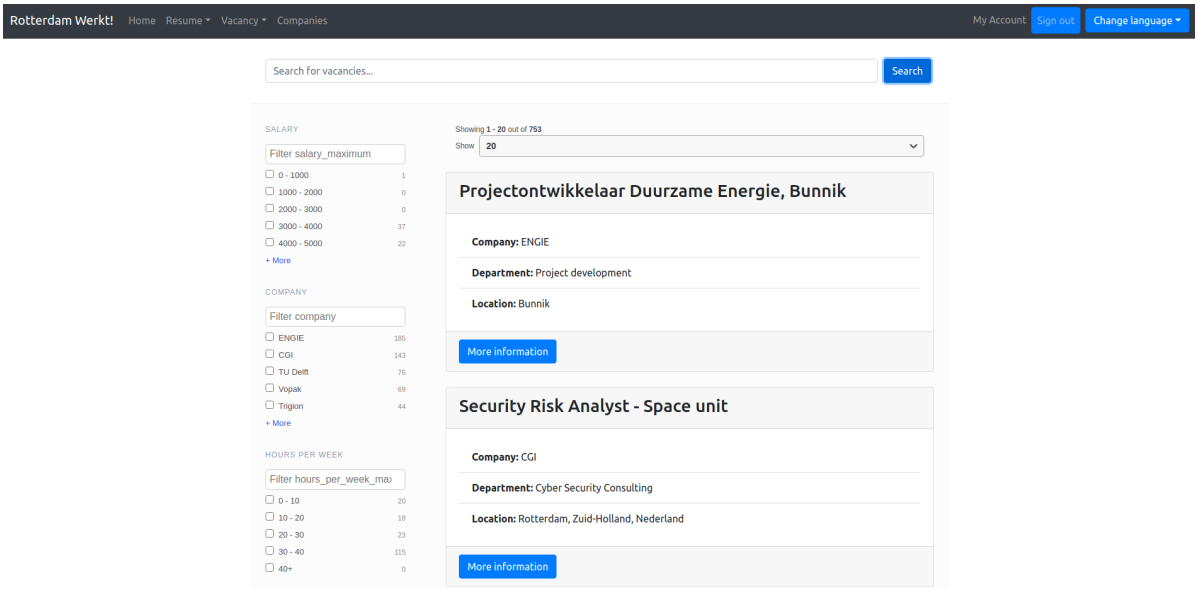
(a) Example vacancy as displayed in the final product



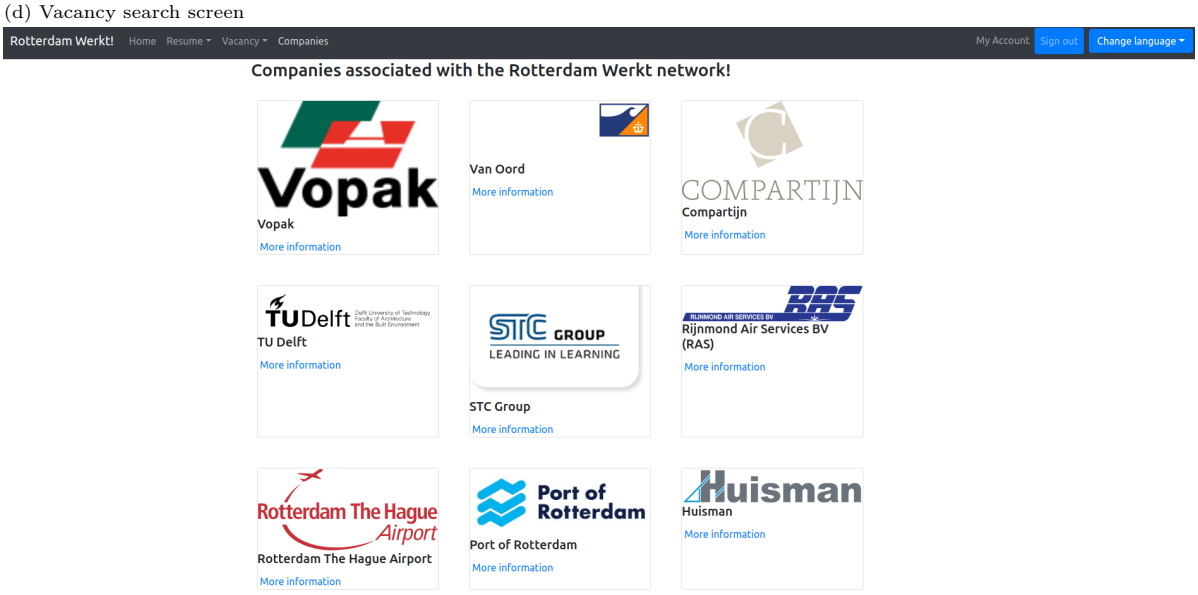
(b) Example resume as displayed in the final product



(c) Example company as displayed in the final product



(d) Vacancy search screen



(e) Page containing all the related companies

Figure 5.1: Screenshots of the final product

5.4. Search engine evaluation

The effectiveness of the search engine is fundamental to the success of the final product, a platform recruiters will use to find vacancies and resumes. As discussed in section 2.4, the search engine is used to filter and retrieve data scraped from websites or inserted by users of the platform, making direct or indirect use of features such as those listed in table 2.4.

In this section, the search engine is evaluated. By evaluating the search engine, it can be measured to what extent the built platform solves the problems defined in section 2.2 from the perspective of the users who will be using the search engine. First, our approach to evaluating the search engine is explicated in section 5.4.1. The search engine's cost and effectiveness are then respectively evaluated in section 5.4.2 and section 5.4.3. Finally, limitations of this evaluation are discussed in section 5.4.4.

5.4.1. Approach

When evaluating a search engine, a distinction is made between *effectiveness*, representing the engine's ability to retrieve relevant documents, *efficiency*, an indication of the speed of the search engine

and of its space requirements, and *cost*, the investments needed to implement and run the search engine. Croft et al. [37] argue that in deciding on the importance and the targets of two of these factors, the third factor is automatically determined. For instance, if effectiveness and cost are deemed most important, and targets are subsequently set for these factors, efficiency is determined by them.

In the context of this research, the importance of these factors is decided by the client. The list of product requirements in table 2.10, derived in cooperation with the client, shows a preference for low development cost and high effectiveness, which may have implications on efficiency. However, considering the client's preference for low cost and high effectiveness over high efficiency, as well as the time constraint of this project, the efficiency of the search engine is not evaluated.

In order to evaluate the costs, we discuss the investments that are required to facilitate the search engine, which are made up of framework, data storage, and maintenance costs. The evaluation of the search engine's effectiveness is based on a comparison of two retrieval functions with a baseline retrieval function and their ability to retrieve relevant documents, after which we explain which retrieval function is used in the platform and why.

5.4.2. Cost evaluation

A requirement of the final product is that all frameworks and programming resources must be free of charge. This requirement was an important factor in the selection of the frameworks and libraries that are used in the final product, as discussed in chapter 2. No frameworks or other programming resources that require purchasing were used in the creation of the platform, and so the framework behind the search engine is free of charge as well, which means the only costs regarding the search engine may come from its maintenance and data storage.

Regarding data storage, the only data that is maintained by the search engine and that takes up an amount of storage of any significance consists of the indices that are used to find data that is stored in a separate database. At the time of writing, the indices concerning scraped vacancy data, with less than 800 vacancies in total, cost less than 11 MB of storage. This can be considered negligible even if the number of vacancies increases by multiple orders of magnitude, which is unlikely as the platform will be used by a mostly constant network of organizations that operate near Rotterdam. Similarly, the logs stored in *Logstash*, as discussed in section 4.5.2, take up little storage: the approximately 35,000 logs accumulated during the development process require around 3 MB of storage. Even considering the number of logs will likely grow more quickly than the amount of vacancies, as logs are generated continuously and are potentially stored for a longer time, the required storage for the logs, too, is negligible. It is therefore not expected that space requirements of the search engine will bring any high costs in the near future.

The cost of maintenance of the search engine is dependent on the future wishes of the client once the platform is transferred to *Rotterdam Werkt!*; any future modifications to the search engine may bring about additional costs. However, if it is only desired that the search engine is kept running without further modifications, we believe this will require minimal effort and thus few costs. We therefore conclude that the search engine is indeed low-cost, which meets the requirements.

5.4.3. Effectiveness evaluation

The effectiveness of a search engine depends on the search engine's ability to retrieve relevant documents. In this research, the retrieved documents are vacancies and resumes. However, we only evaluate the search engine's ability to find relevant vacancies, and not resumes, for two reasons. The first is that resumes require manual input into the platform and are not scraped. Consequently, during the research, we have had access to 677 vacancies posted on the websites of the companies in *Rotterdam Werkt!*, but not to any resumes posted by actual job candidates, that would be used in a practical scenario; there would be no resumes to retrieve and therefore no resume retrieval to evaluate. Secondly, as a part of the problem statement in section 2.2, vacancy retrieval is more fundamental to the platform than resume retrieval in the context of this research.

To be able to measure a vacancy's relevance, we first define what it means for a vacancy to be relevant. We then describe the process of labeling vacancies' relevance through relevance judgments, and what role *search topics* and *pooling* play in this. Lastly, we compare two retrieval functions with a baseline retrieval function and indicate which one looks to be most able to retrieve relevant vacancies.

Definition of relevance
When the platform produced in this research is being used by recruiters, the recruiters' goal is to find vacancies that may suit a certain candidate based on the candidate's resume and future job wishes, and to ultimately propose such vacancies to the candidate so the candidate may find a job they are looking for. As such, a vacancy may be relevant if its demands match a candidate's experience and certificates, and if its topic matches a candidate's wish to be employed in some field. However, although a vacancy's demands may be easily checked against a candidate's background to filter relevant vacancies, they do not necessarily form a rigid bound to the set of relevant vacancies. For instance, a candidate with an education level that is lower than the stated requirement for a given vacancy may still be accepted if they are willing to attend additional schooling to meet the educational requirements later. Similarly, a candidate's wishes to work in some field do not necessarily exclude vacancies in other fields.

This implies that a vacancy's relevance is not as much related to topical relevance as it is to *situational* or practical relevance: the recruiter is interested in any vacancy that may be of practical use to a candidate [38]. Determining whether a vacancy is relevant is thus a subjective process and ideally determined by the future users of the platform (the recruiters looking for relevant vacancies for their candidates). Unfortunately, due to a lack of responses from the recruiters in the network of *Rotterdam Werkt!* to an inquiry about what they find to be relevant vacancies, we had to determine relevance on our own. The implications hereof on the evaluation of the search engine are discussed in section 5.4.4.

Search topics
When users have some information need which they wish to have answered through a search function, they use search queries as a means to retrieve documents that are expected to be relevant to this information need. It therefore makes sense to judge the relevance of retrieved documents relative to this information need. To concretize recruiters' information needs with regard to finding vacancies that match candidates' experience and wishes, we compiled a set of 42 *search topics* that reflect situations in which recruiters have such information needs. Five of these topics were written by recruiters; the remaining 37 were based on postings by recruiters of the network of *Rotterdam Werkt!* on *LinkedIn*. Examples of such topics are as follows[1]:

> "I am looking for a vacancy for an administrative employee with an MBO-level financial background."
> "I am looking for a vacancy for an interim business controller for a retail organization."
> "I am looking for a vacancy for a tax accounting specialist with experience in international structures."

To find vacancies that are expected to be relevant to the search topics, each topic is accompanied by a search query that is to be entered into the search function. The search topics along with their respective search queries are listed in appendix F.

Pooling
Now that we have a set of search topics that reflect recruiters' information needs and a set of retrievable documents consisting of 677 vacancies, the relevance of the vacancies with respect to the search topics can be determined. However, judging for each search topic whether each vacancy is relevant or not would mean performing a relevance judgment over 28,000 times[2] – an infeasible amount given the time constraint of this research and the limited size of the research group. Instead, we make use of *pooling*, where the union is taken of the top-*n* vacancy results of each retrieval function for each search topic [37]. Choosing $n = 10$ would result in at most 30 relevance judgments per search topic when comparing three retrieval functions; all vacancies outside the pools are assumed to be non-relevant with respect to a search topic [39]. We indeed applied pooling with $n = 10$ for the results returned by three retrieval functions for each of the 42 search topics.

Retrieval functions and baseline
As stated in section 2.4, Elasticsearch is the search engine used in the platform. Elasticsearch's default document scoring mechanism uses *BM25*[3] [41], which is also the mechanism that is used in the three

[1] Translated from Dutch.
[2] One judgment per vacancy per search topic, which is $677 * 42 = 28434$.
[3] For more information on BM25, see for instance [40].

evaluated retrieval functions. The evaluated retrieval functions differ in the weights they assign to specific vacancy fields (see figure 3.2). Table 5.2 shows the weights used in the three retrieval functions (RFs).

RF	title	department	education_level	full_text	job_type
Baseline RF	1	1	1	1	1
RF1	2	2	2	1	2
RF2	5	3	4	0.3	2

RF	job_level	job_location	certificate_name	company_name
Baseline RF	1	1	1	1
RF1	2	1	2	1
RF2	3	3	2	1

Table 5.2: Retrieval functions and their weights assigned to vacancy fields

The *baseline* retrieval function serves as a neutral retrieval function against which the other retrieval functions can be compared. All its weights are equal, which means all fields contribute equally to the ranking of vacancies[4].

RF1 and *RF2* have different weights, with a higher weight meaning that the field contributes more to a vacancy's ranking in search results. These weights are based on our expectations, backed by correspondence with recruiters in the network of *Rotterdam Werkt!*, of which fields may be more important to recruiters than others in the context of inputting words in a search query to find particular vacancies. For instance, for both RF1 and RF2, the assumption is made that a vacancy's text description (`full_text`) generally contains more words that are unrelated to the job itself that the vacancy is describing, compared to the other fields, as the full text description may not only contain information about a job, but also about matters such as the company atmosphere or the job's reachability via public transport; and that this vouches for a lower weight for the `full_text` field in the ranking of vacancies.

Relevance judgments and judgment reliability

After pooling, we judged the retrieved vacancies per search topic. Again, it would have been ideal if this was done by recruiters since they would know which vacancies would be relevant to their candidates better than we do, but as was indicated earlier, due to a lack of response to inquiries, we had to judge vacancy relevance ourselves. To make sure we would judge relevance in approximately the same way, we held a calibration session in which we individually judged all vacancies retrieved for a single search topic and then discussed each of these vacancies until we reached a consensus about their judgments.

To indicate what the impact on the results may have been of us judging vacancy relevance ourselves, we distributed the retrieved vacancies of eight search queries to two group members each (for a total of 115 vacancies, and thus 230 combined relevance judgments), with a different pair of members for every two queries, so that an *inter-annotator agreement*, a measure of the agreement among judges (in this case, judges of vacancy relevance), could be calculated [42]. The extent to which judges agree reveals how easy or hard the judgment problem was, which may implicitly indicate the judgments' reliability. We used *Krippendorff's alpha (α)* as a measure for the inter-annotator agreement, as this measure allows for the assessment of more than two judges, takes into account missing data[5], and has software implementations to efficiently calculate the measure[6] [42].

[4] It should be noted, however, that not all companies use the same structure for their vacancy postings, and thus that not all vacancies have information in all vacancy fields; for instance, if a vacancy is scraped that has no information on the job level, the `job_level` field will be empty. A vacancy with a filled-in field of which the contents have a match with a search query will have a higher score compared to an equal vacancy that does not have a value for this field. This means that vacancies with more filled-in fields that match with a search query will be ranked higher compared to vacancies with fewer filled-in fields.

[5] As stated, each of the distributed vacancies was judged by two group members, meaning that there are missing judgments for the other three members for each vacancy.

[6] We used the kripp.alpha function in R's irr library to calculate Krippendorff's alpha based on our relevance judgments. See https://rdrr.io/cran/irr/man/kripp.alpha.html.

The possible values for α range from 0 to 1 (inclusive), where $\alpha = 0$ implies total absence of agreement, and where $\alpha = 1$ implies complete agreement [43]. Based on our relevance judgments, we found a value for α of 0.89, indicating that most judgments by different judges were in agreement with each other. This implies that the fact that the relevance of the vacancies was judged by five different judges, does not reduce the reliability of the relevance judgments – yet it should still be kept in mind that the judgments were not performed by future users (recruiters) who have practical experience with vacancies and matching them to candidates.

The retrieved vacancies of the 34 remaining search queries that were not already distributed, were divided among the group members such that the retrieved vacancies of each remaining query would be judged by one group member.

Effectiveness measures of retrieval functions

Having judged the relevance of all retrieved vacancies for each search topic, we calculated effectiveness measures for each retrieval function in table 5.2. These measures make use of another measure called the *precision*, which indicates for a particular search query the proportion of retrieved vacancies that were judged as relevant [37].

The first calculated measure is the *average precision at rank* r *(*p_r*)*, which indicates the proportion of the top-r retrieved vacancies per search query that are judged as relevant. This is a value between zero and one (inclusive) where zero indicates that none of the first r top-ranked retrieved documents were relevant, and where one indicates that all of them were relevant, on average. The second measure is the *Mean Average Precision (MAP)*, which indicates the average precision for each search result ranking (from one to ten, in this case) for each search query, combined [37]. Table 5.3 shows the values[7] for the average precision at ranks 5 and 10 and the MAP for each retrieval function.

Retrieval function	Precision p_r at rank r $(0 \leq p_r \leq 1)$		MAP $(0 \leq MAP \leq 1)$
	$r = 5$	$r = 10$	
Baseline	0.36	0.41	0.49
RF1	0.38	0.41	0.47
RF2	0.37	0.40	0.46

Table 5.3: Retrieval functions and their corresponding values of the precision at rank 10 and MAP measurements, rounded to two decimals, based on search results of 42 search queries

Comparison of retrieval functions

To be able to decide what the effectiveness measures in table 5.3, and their differences, mean, we conduct *significance tests*. These tests show whether RF1 or RF2 – the *alternative hypotheses* – may be more effective than (reject) the baseline (the *null hypothesis*) [37]. In particular, we will compare the retrieval functions using a *one-tailed paired t-test*[8], which takes into account all the effectiveness measures of all queries individually.

The t-test thus produces a t-value for each different effectiveness measure. The t-values have a corresponding P-value, the probability that the baseline algorithm is capable of producing at least the same t-value, with $0 \leq P \leq 1$. The lower the value for P, the higher the chance that the alternative hypothesis (in this case, RF1 or RF2) is more effective than the baseline; this is typically considered to hold if $P \leq 0.05$ [37]. Table 5.4 shows the results of applying the t-test to the baseline retrieval function and RF1 and RF2.

[7] The values were calculated using trec_eval by the American NIST's Text REtrieval Conference (TREC), downloadable at https://trec.nist.gov/trec_eval/. For more information, see [44].

[8] The paired t-test is defined in [37] as

$$t = \frac{\overline{B - A}}{\sigma_{B-A}}\sqrt{N}$$

where A represents the baseline retrieval function, B the retrieval function to be compared to the baseline, $\overline{B - A}$ the mean of the differences of the effectiveness measures, σ_{B-A} the standard deviation of the differences of the effectiveness measures, N the number of queries used to evaluate the retrieval functions, which is 42, and t is the t-test value.

Retrieval function	Effectiveness measure	t	P
RF1	p_5	0.61	0.27
RF1	p_{10}	-0.13	0.55
RF1	MAP	-0.52	0.70
RF2	p_5	0.27	0.39
RF2	p_{10}	-0.38	0.65
RF2	MAP	-0.64	0.74

Table 5.4: Values for t and P for each retrieval function and each effectiveness measure shown in table 5.3.

Neither RF1 nor RF2 have a P-value lower than 0.27 for any effectiveness measure, meaning they are not likely to be more effective than the baseline retrieval function where all weights for the vacancy fields are set to 1.

Conclusion

Based on the t-test, the baseline retrieval function is not likely to be outperformed by RF1 or RF2, which means the baseline is probably the most effective for practical use at the moment.

5.4.4. Limitations

There are some limitations to this evaluation of the search engine, which we will discuss here. In section 7.2, improvements will be suggested for in future work.

The queries and relevance judgments used for the evaluation of the search engine were created by us. This was done due to a lack of response in to inquiry that was sent out to the recruiters of *Rotterdam Werkt!*. Although we attempted to gather search topics and corresponding search queries that would represent situations that recruiters may encounter in their work, the evaluation would have been more representative of practical use if there had been more input by recruiters. Furthermore, as we had to judge relevance for search results ourselves, the evaluation may not be as reliable as it could have been if recruiters had done the relevance judgments instead.

The retrieval function weights shown in table 5.2 were based on our expectation that some fields may be more important to recruiters than others, in the context of finding vacancies by inputting search queries in the search engine. However, weights could be generated more systematically by separating the vacancy data in a training set and a test set, and where the training set would be used to find out fitting weights and the test set to validate these weights [37].

Lastly, the effectiveness measures discussed in section 5.4.3 (precision at rank 10 and MAP) may provide some insight in the effectiveness of the different retrieval functions, but this insight could be made more complete by providing more different effectiveness measures. Furthermore, the measures are calculated based on a limited amount of vacancy data and search queries, which may have an impact on their accuracy.

5.5. Software Improvement Group

As part of the project, the code base was submitted to the Software Improvement Group[9] (SIG). They analysed the code base with respect to maintainability and reported on seven metrics, all scored out of five stars. Their analysis will be discussed in section 5.5.1 together with the improvements that have been made based on their report.

5.5.1. First submission

Figure 5.2 shows the general analysis performed by SIG. From this analysis it is visible that the system requires some improvements in the aspects of duplication, unit size and unit complexity. Moreover we also received feedback regarding unit interfacing and unit complexity, however no improvements will be made to these as these already have high ratings.

[9] https://www.softwareimprovementgroup.com/

Figure 5.2: SIG general analysis

Duplication

Figure 5.3 shows a table of refactoring candidates proposed by SIG which suffer from duplication. In this table, most of the candidates are composed of React classes, in particular the screens and forms. Further investigation has shown that there is indeed a lot of overlap in the way we use the 'Formik'[10] library to create forms for users to add and edit data.

In order to overcome this problem, 'Formik' provides a hook, `useFormik()` [33] which allows for easy abstraction of form creation. This hook returns an object which contains the state of the form and methods to update the state. This allowed for the form fields to be more generalized into a `FormGroup` component which uses this object in order to render an input element on the screen.

This also meant that boiler plate code such as error handling and form submitting could be refactored away into the abstracted method since this is very similar to all forms.

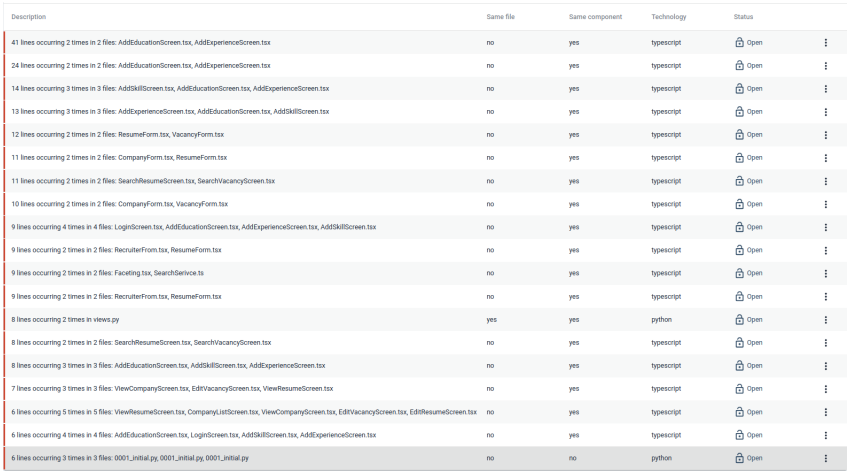


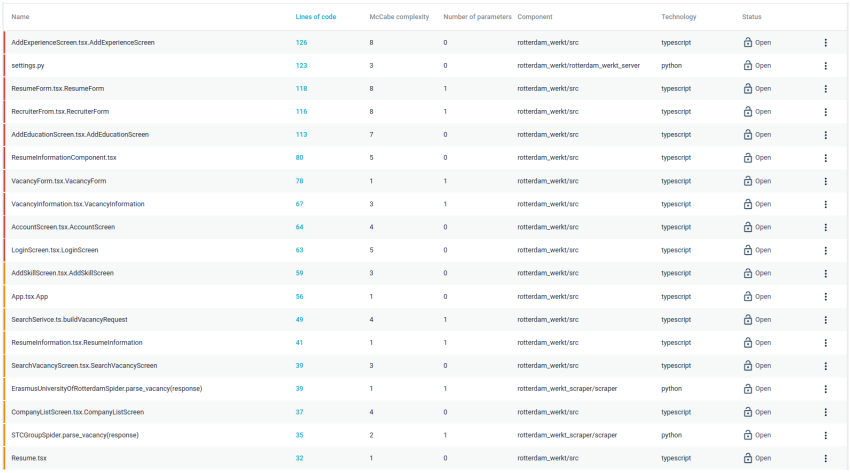
Figure 5.3: SIG duplication refactor candidates

Unit size

Figure 5.4 shows a table of refactoring candidates proposed by SIG which are considered to be too large. At first sight there seems to be a significant overlap between the unit-size refactoring candidates and the duplication candidates. This suggests that when we perform our abstraction from section 5.5.1, many of these unit-size problems will be solved since the duplicate code will be removed.

Nonetheless there are still some units which remain after this refactor, most notably the `settings.py` and `Resume.tsx`. For the `settings.py` file there are limited options, given that this is the settings file the Django project expects and thus all settings must be placed in here therefore it

[10] https://formik.org/



Name	Lines of code	McCabe complexity	Number of parameters	Component	Technology	Status
AddExperienceScreen.tsx.AddExperienceScreen	126	8	0	rotterdam_werkt/src	typescript	Open
settings.py	123	3	0	rotterdam_werkt/rotterdam_werkt_server	python	Open
ResumeForm.tsx.ResumeForm	118	8	1	rotterdam_werkt/src	typescript	Open
RecruiterFrom.tsx.RecruiterForm	116	8	1	rotterdam_werkt/src	typescript	Open
AddEducationScreen.tsx.AddEducationScreen	113	7	0	rotterdam_werkt/src	typescript	Open
ResumeInformationComponent.tsx	80	5	0	rotterdam_werkt/src	typescript	Open
VacancyForm.tsx.VacancyForm	78	1	1	rotterdam_werkt/src	typescript	Open
VacancyInformation.tsx.VacancyInformation	67	3	1	rotterdam_werkt/src	typescript	Open
AccountScreen.tsx.AccountScreen	64	4	0	rotterdam_werkt/src	typescript	Open
LoginScreen.tsx.LoginScreen	63	5	0	rotterdam_werkt/src	typescript	Open
AddSkillScreen.tsx.AddSkillScreen	59	3	0	rotterdam_werkt/src	typescript	Open
App.tsx.App	56	1	0	rotterdam_werkt/src	typescript	Open
SearchService.ts.buildVacancyRequest	49	4	1	rotterdam_werkt/src	typescript	Open
ResumeInformation.tsx.ResumeInformation	41	1	1	rotterdam_werkt/src	typescript	Open
SearchVacancyScreen.tsx.SearchVacancyScreen	39	3	0	rotterdam_werkt/src	typescript	Open
ErasmusUniversityOfRotterdamSpider.parse_vacancy(response)	39	1	1	rotterdam_werkt_scraper/scraper	python	Open
CompanyListScreen.tsx.CompanyListScreen	37	4	0	rotterdam_werkt/src	typescript	Open
STCGroupSpider.parse_vacancy(response)	35	2	1	rotterdam_werkt_scraper/scraper	python	Open
Resume.tsx	32	1	0	rotterdam_werkt/src	typescript	Open

Figure 5.4: SIG unit size refactor candidates

was decided to not refactor this file. Furthermore, the `Resume.tsx` file could be a false positive for this metric. The file consists of several small interfaces, which are used by the TypeScript typing system. Also for this file it was decided to not refactor it, given that it actually exists out of 10 interfaces (of which only eight are counted towards the unit size) which are each at most ten lines. The only option would be to further split these interfaces into different files, however this would also increase the number of files in the project and thus still make it more complex.

Additionally, from the scraping point of view, the SIG report mentions several spiders as possible candidates for refactoring. However, the suggested methods and functions were relatively long for valid reasons. The common characteristic amongst the suggested candidates is the fact that the candidate methods assigned values to a large number of vacancy fields. That is, the spiders the methods belong to retrieve a lot of data. The data assignment had to be centralized in a single method to keep the methods simple, coherent and readable. If more than, or close to, ten fields were retrieved, the method contains one line per field, sometimes a few lines more. Splitting the method into smaller methods would therefore have little effect and only needlessly increase complexity and decrease readability, as each field has to be assigned at some point. Another common factor was nested methods, which increase the unit length of the outer function or method. Given that nested methods can simply be unnested, at the cost of sacrificing enclosed scopes, the unit length could be reduced. This has been done for all suggested candidates in the scraper.

Unit complexity

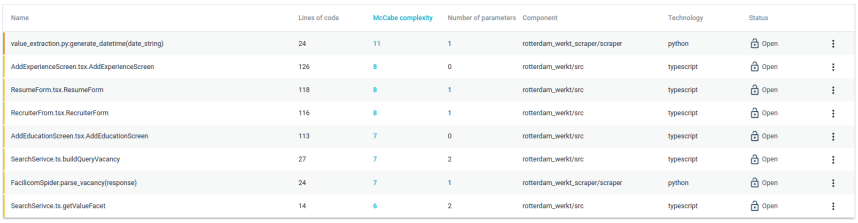
Figure 5.5 shows a table of refactoring candidates proposed by SIG which are considered to have a McCabe complexity which is too high. This is a complexity which is based on a graph representation of a method where statements such as `if` and `for` can cause branching [45]. Keeping the complexity low has the advantage that the method needs to be tested less extensively, since there are less cases to test and the code becomes more readable.

This list shows that there is reasonable overlap with the Duplicate code and Unit size. Given that these files have already been abstracted, many of the `if` and `for` statements have also been abstracted away. This results in having more units, but each unit being less complex.

The SIG report also suggests that a couple of methods used in the scraper could be reduced. The two main candidates are the `generate_datetime` function and the `parse_vacancy` method for several spiders. The common factor for these candidates is the fact that the methods make use of nested methods or functions, which increase code complexity as well as unit size. After careful consideration, the nested methods and functions have been unnested, since we believe this to improve readability, decrease complexity while sacrificing the enclosed scope of said methods and functions.

Unit interfacing

The SIG report states that the scraper contains several methods ought to be refactored with regards to the number of parameters. Though, four out of five candidates are methods of the framework used.



Name	Lines of code	McCabe complexity	Number of parameters	Component	Technology	Status
value_extraction.py:generate_datetime(date_string)	24	11	1	rotterdam_werkt_scraper/scraper	python	Open
AddExperienceScreen.tsx.AddExperienceScreen	126	8	0	rotterdam_werkt/src	typescript	Open
ResumeForm.tsx.ResumeForm	118	8	1	rotterdam_werkt/src	typescript	Open
RecruiterFrom.tsx.RecruiterForm	116	8	1	rotterdam_werkt/src	typescript	Open
AddEducationScreen.tsx.AddEducationScreen	113	7	0	rotterdam_werkt/src	typescript	Open
SearchSerivce.ts.buildQueryVacancy	27	7	2	rotterdam_werkt/src	typescript	Open
FacilicomSpider.parse_vacancy(response)	24	7	1	rotterdam_werkt_scraper/scraper	python	Open
SearchSerivce.ts.getValueFacet	14	6	2	rotterdam_werkt/src	typescript	Open

Figure 5.5: SIG unit complexity refactor candidates

Thus, those candidates need not be adjusted. The fifth candidate, on the other hand, namely `full_text_sanitizer` in `sanitizer.py`, has not been refactored either, as the `str` parameter does is not present in the function and the parameters `outer_joining_string` and `inner_joining_string` have default values, which only need to be specified if the callee wishes to deviate from this default, this is only needed in very few cases.

Module coupling
According to the SIG report, these two scripts have high coupling: `sanitizer.py` and `value_extraction.py`. Though, we were already aware that this would be the case. The two suggested files contain functions that are crucial for handling and extracting data. As each spider needs data sanitization and data extraction, the scripts are called often, hence it was decided not to make any refactors.

5.6. Ethical implications

Considering that data is being logged, ethical implications ought to be taken into account. Besides the fact that GDPR compliance has become a must in recent years, being able to remove someone's data when requested is also an ethical must. Extra caution has therefore been taken such that we are not logging personal data, or data which can be linked to a person. This means that personal data will only be stored in a database and Elasticsearch of which both data can easily be removed. The database can easily remove a single entry linking to a person and Elasticsearch can delete a complete index and recreate it from the database, after the user was deleted, within minutes.

6

Process evaluation

During the project, some of the choices made regarding the process turned out to work really well whilst others did not seem to work or resulted in a different outcome. In this section a short description will be given of the positive and respectively the negative parts of our process, with ways to improve or avoid them in future projects.

There were quite a few process choices that positively affected our project. The development methodology, Scrum, was very helpful in being able to break down the project in smaller pieces and setting periodic goals for the project. These goals made it possible for everybody in the team to have clear targets while working on the project. The daily stand-ups allowed the entire team to be continuously up to speed. It encouraged each team member to actively discuss their current tasks and any potential struggles. Whenever anybody pointed out any difficulties, others were always able to jump in and help out in order to overcome these hurdles. This significantly sped up the development time as nobody could spend more than one day stuck on a certain issue.

Nonetheless, there were also some issues in the development process. In our initial planning we expected that implementing the scraper would be done by week five of the project. Unfortunately, creating the scraper was more complex in the end and therefore took more time. In the end the scraper was finished in week eight. This meant less development time was put into the front-end and the back-end, hence some of the should-have and could-have features have not yet been finished. For future projects it is very important to take this complexity in mind, and to be very mindful of the impact that a reduced amount of the development power could have on the final product.

Furthermore, it would have been desirable to have kept the client more involved throughout our development process. However, this project lasted for roughly ten weeks, of which several weeks were spent on doing research and writing the report. For an organization that has existed since 2016[46], ten weeks is relatively insignificant. Given that said organization comprises several companies and organizations, each with their own desires and priorities, one may consider our main client to actually be several smaller clients. As a result of these factors, scheduling a meeting was rather difficult. It often took over a week to plan a meeting, which is a relatively substantial amount of valuable time.

Despite these challenges, we tried to involve the client in the most important phases of this project: setting the requirements and evaluating the platform. While we were able to successfully involve the client in the former phase, as we received valuable feedback, the latter phase posed some impactful issues. In the latter phase, we asked the client to provide feedback for the platform we created, by providing search topics that we could use for evaluating the search engine. We believe to have given them sufficient time to do so, namely at most four hours of work spread over roughly ten working days. However, either the time we provided was insufficient, or the client considered the evaluation to be of lesser priority. Either way, this meant we had to create search topics and queries ourselves, which cost significant time and has possibly resulted in a less accurate search engine. One of the most challenging aspect of this projects was therefore cooperating with more than just one client within such a short time.

7

Conclusion and future work

7.1. Conclusion

At the start of this project, *Rotterdam Werkt!*'s wish was to have a platform which combines all vacancies from all the *Rotterdam Werkt!* organizations. This platform helps their recruiters to find new positions for their current employees in order to boost internal mobility of the work force. Even though platforms such as Indeed and LinkedIn already exist, these are all paying services and they wanted a private non-subscription platform.

At the start of the project a lot of research was done into similar platforms and potential tools that could be used in order to make this project successful. The two major tools our project depends on are Scrapy for collecting vacancies from the companies related to *Rotterdam Werkt!*, and Elasticsearch which provides a RESTful search engine such that the user can search and filter through the vacancies which were scraped.

After two weeks of research and seven weeks of implementation, the system was fully designed, implemented and tested. This includes an evaluation of the performance of the search engine. However limited due to the lack of data, this evaluation provided important insights into the future development and optimization of the search engine in order to further improve the platform.

To conclude, we believe this project was a success. We have created a tool which will collect all the vacancies from each website of the organizations in *Rotterdam Werkt!* and provided a platform which will allow the recruiters to search for the vacancies and resumes. Furthermore, the tools have been provided in order to collect more data for a future evaluation and optimization of the search engine which is strongly advised. Lastly, considering that our system was developed and tested in 7 weeks proves the power of the usage of modern and suitable frameworks, therefore we believe that our system is easily extensible in order to allow for future changes in the *Rotterdam Werkt!* network.

7.2. Future work and recommendations

In the previous sections, some limitations to the current system have been described. Based on these limitations, some suggestions and future work options will be given. In section 7.2.1 future work for the search engine is discussed, future work for query logging in section 7.2.2, and in section 7.2.3, future work for the scraper is discussed.

7.2.1. Search engine evaluation

In section 5.4, the evaluation of the search engine was discussed. This evaluation has some limitations as mentioned in section 5.4.4. One limitation is that search topics and search queries were mostly derived by us from posts on *LinkedIn* by recruiters, and that most were thus not directly provided to us by recruiters for the sake of the evaluation of the final product. Ideally, the users of the search engine (the recruiters, in this case) would provide a list of search topics with accompanying search queries for the evaluation of the search engine, as recruiters have practical experience with finding particular vacancies and matching vacancies with potential job candidates. Furthermore, although it was shown in section 5.4.3 that the inter-annotator agreement among the group members performing relevance

judgments was relatively high, the relevance judgments may be more reliable if they would be done by recruiters themselves, since, again, they have more experience in judging whether some vacancy suits a job candidate or not.

We therefore recommend to repeat this evaluation with search topics, search queries and relevance judgments that are provided directly by recruiters. In order to aid this, a query logging system has been implemented. This will record all the queries performed in the application. Being able to log these queries helps with gathering queries for a new evaluation. Since these queries are obtained in the production phase from actual users of the system, it can be expected that they will be more relevant for the evaluation than the ones which were used in this research. Methods to apply query logging in the evaluation of a search engine are discussed in [37].

Furthermore, we also suggest to extend this evaluation to the resume search engine. Due to a lack of resume data and insufficient time to gather more data, it was not possible to conduct a significant evaluation. By gathering resume data, search topics with accompanying search queries concerning resume searching, as well as relevance judgments, the resume search engine can be evaluated and subsequently optimized as well.

Regarding the retrieval functions that were compared in section 5.4.3, the weights could be generated more systematically. A proposal is to separate vacancy and resume data in training and test sets, and to use the training sets to find out fitting weights for document attribute fields, and the test sets to validate these weights [37]. This way, the retrieval functions' weights can be tailored especially to the tasks of finding relevant vacancies and resumes.

Lastly, the effectiveness measures calculated in section 5.4.3 are based on a limited amount of vacancy data and search queries. Their accuracy, and thus that of the final comparison of retrieval functions, can be improved by evaluating the search engine based on more vacancy data and search queries.

7.2.2. Query logging

Currently queries are logged at a very verbose level as can be seen in figure 4.3. Only exact queries can be matched, meaning that the filters, search term and size must match in the retrieval function before the queries will be identified as equal. In order to better perform query logging it would be valuable to extract the filter, search term and size. This means that more aggregations can be applied to the data in order to further identify potential trends. This will furthermore improve the search engine evaluation described in section 7.2.1 as more constructive data will be available to use.

7.2.3. Scrapers

Even though the scraper is capable of scraping the vacancy websites of all organizations connected to *Rotterdam Werkt!* containing vacancies, the scraper can be improved. For instance, the scraper is tailored to the current vacancy websites. If these websites change significantly, either their layouts, URLs or architecture, the scraper will fail. Future work thus lies in making the scraper more robust such that it is able to adapt to such changes to larger extent. Additionally, if more organizations join *Rotterdam Werkt!* and if said organizations wish to be included in the centralized vacancy platform, the scraper should be extended manually. That is, a separate spider should be implemented for that organization.

Appendices

A

Original problem statement

The content in this appendix is a direct copy of the original BEP proposal published on ProjectForum (https://projectforum.tudelft.nl/). Although the text remains identical, small changes have been made to the layout to reflect the visual style of this document.

A.1. Project description

“Rotterdam Werkt!” is een netwerk van ongeveer 20 organisaties in de regio Rotterdam. De TUD en de EUR zijn er lid van. Het netwerk is opgezet om onderlinge mobiliteit van medewerkers te vergroten. Elke organisatie heeft zijn eigen webpagina’s voor de publicatie van vacatures en tijdelijke projecten. Het nu voorgestelde project moet er toe leiden dat er een search-tool komt waarmee de mobiliteitsofficers / HR-adviseurs maar ook alle medewerkers van de bij het netwerk aangesloten bedrijven een actuele listing onder ogen krijgen van alle op de ongeveer 20 sites van de bij het netwerk aangesloten organisaties gepubliceerde vacatures.
De search-tool moet daartoe de betreffende pagina’s van de aangesloten organisaties dagelijks scannen en de daarop geplaatste vacatures op een logische wijze rubriceren. De mobiliteitsmanagers, HR-adviseurs en medewerkers moeten gericht naar specifieke vacatures en naar specifieke kenmerken daarvan kunnen zoeken.

B

Info sheet

The following page contains the info sheet used to provide a short overview of the client company, the project and the team members.

The remainder of this page is intentionally left blank.

Title of the project: Improving interorganizational mobility through centralizing vacancies
Name of the client organization: *Rotterdam Werkt!*
Date of final presentation: 29th January 2021

Description

Rotterdam Werkt! is a network of 14 companies in the region of Rotterdam. The network was setup in order to increase employee mobility. Each of these organizations have their own website with vacancies and temporary projects. Currently the recruiters of each company have to search through each website individually in order to find a vacancy for one of their current employees inside the network.

The main challenge during this project was to retrieve all the vacancies which belong to companies associated to the network and allowing these vacancies to be inserted into a search-tool such that recruiters can easily find vacancies for their employees. Since the websites of the associated companies may change and companies may join the network, the challenge here was to make sure that this scraper is maintainable. Furthermore, since the network is planning on using other students to maintain this project, the rest of the code base also needs to be maintainable and easily deployable.

During the research phase the main focus was on finding the best tools in order build the scraper and the search engine. As a result, it was decided to use Elasticsearch as a search engine and Scrapy for the scraper.

The final product includes two server-side components, the scraper which will retrieve all the vacancies and the API which is responsible for storing vacancies, authentication and authorization. Furthermore, it also includes a front-end which is connected to Elasticsearch and the API in order to allow users to interact with the data. The final product could be developed further in order to include all the could-have and should-have features. Also, the search engine could be optimized further as more data becomes available for this.

Members

L.E. van Hal masprojectl0l0l0l@gmail.com
Interests: Software engineering, (software) languages
Contributions: Scraping implementation, client contact, search engine evaluation
H.A.B. Janse hendrikjanse@live.com
Interests: Software engineering, programming languages, static code analysis
Contributions: Back-end implementation, API-implementation, Serializer-implementation, Database Implementation
D.R. den Ouden nyetvoorspam@outlook.com
Interests: Software engineering, databases, scripting & automation
Contributions: Scraping implementation, CI pipeline and quality assurance
R.H. Piepenbrink rolfpiep@outlook.com
Interests: Software engineering
Contributions: Scraping implementation and quality assurance
C.S. Willekens cedric.willekens@hotmail.com
Interests: Software engineering, Programming languages, Static code analysis, Software architectures
Contributions: Front-end implementation, Back-end implementation, API implementation, Elk-Stack setup, Logging setup, Deploy setup

Contact Information

Client	H. Bolk	bolk@law.eur.nl
	R. Rotmans	r.rotmans@portofrotterdam.com
Coach	Dr. C. Hauff	c.hauff@tudelft.nl

The final report for this project can be found on https://repository.tudelft.nl

C

Rotterdam Werkt! organizations

Company name	Vacancies overview page URL
Rotterdam Ahoy	https://www.ahoy.nl/informatie/vacatures
TU Delft	https://www.tudelft.nl/en/about-tu-delft/working-at-tu-delft/search-jobs/
Rotterdam The Hague Airport	https://www.werkenoprotterdamthehagueairport.nl/vacatures
KOTUG	https://www.kotug.com/about/careers
Vopak	https://www.careersatvopak.com/vacancies
Erasmus University of Rotterdam	https://www.eur.nl/en/working/vacancies-academic-staff, https://www.eur.nl/en/working/vacancies-support-staff
CGI	https://www.cginederland.nl/nl/careers-search?country_id=469
Evides	https://www.werkenbijevides.nl/
Vanoord	https://www.vanoord.com/nl/werken-bij/vacatures
STC Group	https://stc-group.nl/vacatures
Port of Rotterdam	https://www.portofrotterdam.com/nl/vacancies/all
Engie	https://jobs.engie.com/netherlands/jobs/search/reference
Facilicom Group	https://werkenbijfacilicom.nl/vacatureoverzicht/
Huisman	https://www.werkenbijhuisman.nl/vacatures

D

Rotterdam Werkt! information sheet

The remainder of this page is intentionally left blank.

Realiseert vrijwillige arbeidsmobiliteit

Wat is Rotterdam Werkt?

Rotterdam Werkt is een netwerk waarin vrijwillige arbeidsmobiliteit mogelijkheden aan worden geboden voor getalenteerde en bevlogen mobiele medewerkers tussen aangesloten organisaties in de regio Rotterdam.

Hoe werkt Rotterdam Werkt?

Het netwerk zoekt naar mogelijkheden en oplossingen om gezamenlijk de arbeidsmobiliteit tussen de aangesloten organisaties te verbeteren, op een geheel vrijwillige basis en waar partijen geen financiële bijdrage hoeven te leveren maar onderling afstemmen over de financiën. Arbeidsmobiliteit wordt bevorderd op de volgende manieren:

- Vacatures – Het delen van vacatures met de aangesloten organisaties
- Detachering – Het “uitlenen” van personeel tussen de aangesloten organisaties
- Uitwisseling – Het uitwisselen van werknemers tussen de aangesloten organisaties
- Het delen van kennis over gerelateerde HR vraagstukken
- Ook kunnen er gezamenlijke projecten opgezet worden (bijv. een traineeship)

Rotterdam Werkt biedt kandidaten zowel tijdelijke als vaste werkervaringsplaatsen voor de gehele werkende populatie (18-65 jaar). Werknemers krijgen hiermee de kans om binnen andere organisaties nieuwe werkervaringen op te doen zodat zij zich beter kunnen oriënteren op ander werk. Daarnaast kunnen werknemers zich door blijven ontwikkelen op de arbeidsmarkt door het leren van nieuwe vaardigheden en competenties. Ook voor de aangesloten organisaties is het een uitkomst om gebruik te kunnen maken van kennis en talent van andere organisaties in de regio.

Wat gebeurt er binnen Rotterdam Werkt?

Vier keer per jaar komen de partners binnen het netwerk bijeen in partnerbijeenkomsten. Daarnaast zijn er vier keer per jaar recruitersbijeenkomsten waar business partners en recruiters van de aangesloten organisaties bijeenkomen. De bijeenkomsten vinden op toerbeurt plaats bij de aangesloten organisaties.
Elke betrokken organisatie uit het netwerk wordt in deze bijeenkomsten vertegenwoordigd door tenminste een HR professional. Deze HR professionals fungeren ook als contactpersoon voor het Rotterdam Werkt netwerk. Naast deze bijeenkomsten, worden openstaande werkervaringsplaatsen (vacatures) en aanbod van personeel continue met elkaar gedeeld in het netwerk, d.m.v. e-mails of persoonlijk contact.

Afspraken binnen Rotterdam Werkt

- Circulaire economie van mensen creëren
- Coallition of the willing
- Open en eerlijke communicatie
- Betrokkenheid
- Resultaatgerichtheid
- Gesloten beurzen

Als aangesloten partner bent u onderdeel van het netwerk en zorgt u voor een actieve bijdrage. Dit doet u door kennis te delen en sterk betrokken te zijn bij het matchen van kandidaten.

E

Elasticsearch mappings

E.1. Vacancy

```
{
  "mappings": {
    "_doc": {
      "properties": {
        "certificate_name": {
          "type": "text"
        },
        "company": {
          "type": "nested",
          "properties": {
            "title": {
              "type": "keyword"
            }
          }
        },
        "company_name": {
          "type": "text"
        },
        "department": {
          "type": "keyword"
        },
        "education_level": {
          "type": "keyword"
        },
        "end_date": {
          "type": "date"
        },
        "full_text": {
          "type": "text",
          "fields": {
            "en": {
              "type": "text",
              "analyzer": "english"
            },
            "nl": {
              "type": "text",
              "analyzer": "dutch"
            }
```

```
            }
          },
          "hours_per_week_maximum": {
            "type": "integer"
          },
          "hours_per_week_minimum": {
            "type": "integer"
          },
          "id": {
            "type": "integer"
          },
          "job_level": {
            "type": "keyword"
          },
          "job_location": {
            "type": "keyword"
          },
          "job_type": {
            "type": "keyword"
          },
          "required_certificates": {
            "type": "nested",
            "properties": {
              "name": {
                "type": "keyword"
              }
            }
          },
          "salary_maximum": {
            "type": "integer"
          },
          "salary_minimum": {
            "type": "integer"
          },
          "title": {
            "type": "text",
            "fields": {
              "en": {
                "type": "text",
                "analyzer": "english"
              },
              "nl": {
                "type": "text",
                "analyzer": "dutch"
              }
            }
          },
          "years_of_experience": {
            "type": "integer"
          }
        }
      }
    }
}
```

Code snippet E.1: Elasticsearch vacancy mapping

E.2. Resume

```
{
  "mappings": {
    "_doc": {
      "properties": {
        "educations": {
          "type": "nested",
          "properties": {
            "degree": {
              "type": "text"
            },
            "description": {
              "type": "text"
            },
            "field_of_study": {
              "type": "text"
            },
            "university": {
              "type": "text"
            }
          }
        },
        "email_address": {
          "type": "text"
        },
        "experiences": {
          "type": "nested",
          "properties": {
            "company": {
              "type": "text"
            },
            "description": {
              "type": "text"
            },
            "employment_type": {
              "type": "text"
            },
            "headline": {
              "type": "text"
            },
            "title": {
              "type": "text"
            }
          }
        },
        "extra_info": {
          "type": "text",
          "fields": {
            "nl": {
              "type": "text",
              "analyzer": "dutch"
            },
            "raw": {
              "type": "text"
            }
```

```
            }
          },
          "first_address_line": {
            "type": "text"
          },
          "first_name": {
            "type": "text"
          },
          "id": {
            "type": "integer"
          },
          "last_name": {
            "type": "text"
          },
          "second_address_line": {
            "type": "text"
          },
          "skills": {
            "type": "nested",
            "properties": {
              "description": {
                "type": "text"
              },
              "name": {
                "type": "text"
              }
            }
          }
        }
      }
    }
}
```

Code snippet E.2: Elasticsearch resume mapping

F

Search topics

Search topic	Search query
Ik zoek een vacature voor...	
een directiesecretaresse die op C-Level heeft gewerkt.	directiesecretaresse c-level
een inkoper met een technische achtergrond (HBO).	inkoper technisch HBO
een assistent assetmanager met een bouwkundige achtergrond (MBO +).	assistant assetmanager bouwkundig
een administratief medewerker met financiele achtergrond (MBO).	administratief medewerker financieel MBO
een communicatieadviseur met kennis van evenementen.	communicatieadviseur evenementen
een professionele HR-generalist die op zoek is naar een uitdagende baan in een dynamisch offshore-bedrijf.	HR offshore
een tweede stuurman op een hopper, voor minstens zes weken.	tweede stuurman hopper
een matroos met multicat-ervaring (rond 15-10 opstappen).	matroos multicat 10 opstappen
een basisarts met geldige BIG-registratie die in de Jeugd GGZ wilt werken.	basisarts BIG jeugd GGZ
een senior HR business partner die wilt werken voor een internationale opdrachtgever uit de omgeving Rotterdam, met een focus op MD en organisatieontwikkeling, voor een salaris rond de 85K.	senior HR organisatieontwikkeling MD internationaal
een communicatiemedewerker met interesse in eventmanagement en notuleren.	communicatiemedewerker eventmanagement notuleren
een toezichthouder in kabelinstallatie voor offshore-projecten.	offshore kabelinstallatie toezichthouder
een interim Business Controller met ervaring in het uitwerken van business cases en het ontwikkelen van dashboards.	business controller dashboards business cases interim
een interim business controller voor een retailorganisatie.	business controller retail interim
een tax accounting specialist (RA) met ervaring met internationale hoofdstructuren.	tax accounting specialist RA internationaal
een elektromonteur met een afgeronde MBO-opleiding binnen de Elektrotechniek met ervaring binnen de industrie.	elektromonteur elektrotechniek industrie
een interim werktuigbouwmonteur met chemische ervaring.	werktuigbouwmonteur chemie interim
een Lead Application Engineer binnen de industriële automatisering met ervaring in het ontwerpen en programmeren van applicatiesoftware voor PLC-, SCADA- en/of DCS-systemen.	lead application engineer industriële automatisering PLC SCADA DCS

een werktuigbouwkundig monteur met interesse in beheer en onderhoud van gebouwgebonden werktuigbouwkundige installaties.	werktuigbouwmonteur beheer onderhoud gebouwgebonden
een front-end developer met ervaring met VueJS, Angular, React, HTML, Javascript, Typescript, CSS, NodeJS, NPM en Git, die wilt werken in een scrumteam.	front-end developer vuejs angular react html javascript typescript css nodejs npm git scrum
een starter in de software development die wilt werken met nieuwe big data en machine learning technologieën en cloudtechnologieën.	starter software development big data machine learning cloud
een Java-ontwikkelaar met veel kennis van Spring en Maven en ervaring met het werken in een Agile-omgeving en scrumprojecten.	java spring maven agile scrum
een afgestudeerde HBO'er of WO'er met affiniteit met IT, die in het bezit is van een Scrummaster-certificaat zoals PSM-I, en met minimaal 3 jaar ervaring in de afgelopen 5 jaar bij grote en complexe organisaties.	IT scrummaster 3 jaar ervaring HBO WO
iemand die deel uit wilt maken van een snelgroeiend SAP S/4HANA-team.	SAP-S 4HANA
fulltime Java developer voor een jaar, met afgeronde relevante HBO-opleiding of WO Bèta-studie, 5 jaar ervaring als Java-ontwikkelaar, 4 jaar ervaring met Spring, SQL, JPA/Hibernate, en Webservices/SOA/WSDL, en 1 jaar ervaring met Wicket.	java developer spring sql jpa hibernate webservices soa wsdl wicket ervaring
interim HR SSC/HR Processes en Payroll Transformation manager met zeer veel ervaring.	HR SSC processes payroll transformation manager ervaring interim
freelance matchmaker met een afgeronde HBO-opleiding, recente ervaring als intercedent of bij een werkgeversservicepunt, en ervaring met Civision en Sonar/WBS, voor 36 uur per week.	matchmaker HBO intercedent werkgeversservicepunt civision sonar WBS freelance
werk- of teamcoach met NOBCO-certificering of TTISI- (bij voorkeur DISC- en/of Driving Forces-)certificering.	werkcoach teamcoach nobco ttisi disc driving forces
een planner met kennis van geotechnisch en geofysisch materiaal (offshore)	planning geotechniek geofysisch materiaal
een logistiek magazijnmedewerker die Engels spreekt	logistiek magazijnmedewerker
een controller met een HBO-opleiding en 5 jaar ervaring in financiële administratie	controller HBO financiën
een junior data analist met een economie- of finance-HBO-opleiding en ervaring met Qlikview	data-analist economie HBO
een assistent in de recruitment (HR) met vaardigheid in social media, een afgeronde HBO-opleiding richting HR of commercie, en maximaal 2 jaar werkervaring	HR recruitment assistant social media HBO commercie 2 jaar werkervaring
een online marketeer, minimaal op HBO-niveau, met 4 jaar ervaring in Google Ads, Facebook Ads Manager, Bing Ads, Excel, Google Tag Manager en Google Analytics, en beheersing van een vreemde taal (Engels/Duits/Frans)	online marketeer HBO 4 jaar ervaring google ads bing ads facebook ads manager excel google tag manager google analytics vreemde taal
een projectmanager met een WO-titel in engineering met minimaal 5 jaar ervaring in industriële projectmanagement, beheersing van de Engelse taal en kennis van Project Management Software	projectmanager WO engineering 5 jaar ervaring industrie Engels project management software

een shipment planner met kennis van Office-programma's, SAP by Design en wegtransport, en beheersing van de Nederlandse en Engelse taal	shipment planner Office SAP by Design wegtransport Nederlands Engels
een techniciën op MBO-4-niveau (E/I) met een VCA/VOL-certificaat en minimaal 2 jaar ervaring in de chemische industrie	techniciën MBO-4 VCA VOL 2 jaar ervaring chemische industrie
een visual designer op HBO-niveau met minimaal 2 jaar werkervaring, en kennis van Adobe Photoshop, Illustrator en Indesign (Creative Suite)	visual designer HBO 2 jaar ervaring adobe photoshop adobe illustrator adobe indesign creative suite
een CV-monteur met een MBO-opleiding Installatietechniek, werkervaring binnen de installatietechniek, en een VCA-certificaat	CV-monteur MBO installatietechniek VCA werkervaring
een Teamleider Logistiek met kennis van LEAN en minimaal 3 jaar werkervaring in een leidinggevende rol in de logistiek	teamleider logistiek LEAN 3 jaar ervaring leidinggevende
een maritiem elektromonteur met rijbewijs B, een VCA-certificaat, en een afgeronde MTS- of MBO-opleiding richting elektrotechniek of mechatronica	maritiem elektromonteur MTS MBO VCA elektrotechniek mechatronica rijbewijs B
een Lead Software Engineer (maritiem) met een afgeronde technische HBO-opleiding, werkervaing met grote projecten in industriële automatisering, en ervaring in het ontwikkelen van PLC-software en SCADA	lead software engineer maritiem HBO technisch industriële automatisering PLC SCADA

Table F.1: All search topics with their corresponding search queries used in the evaluation of the search engine.

G

Logged queries

```
{
   ”size”:0,
   ”query”:{
      ”bool”:{
         ”should”:[
            {
               ”match_all”:{
                  ”boost”:1.0
               }
            }
         ],
         ”adjust_pure_negative”:true,
         ”boost”:1.0
      }
   },
   ”aggregations”:{
      ”certificates”:{
         ”nested”:{
            ”path”:”required_certificates”
         },
         ”aggregations”:{
            ”names”:{
               ”terms”:{
                  ”field”:”required_certificates.name”,
                  ”size”:2147483647,
                  ”min_doc_count”:1,
                  ”shard_min_doc_count”:0,
                  ”show_term_doc_count_error”:false,
                  ”order”:[
                     {
                        ”_count”:”desc”
                     },
                     {
                        ”_key”:”asc”
                     }
                  ]
               }
            }
         }
      }
```

```
    },
    "highlight":{
        "fragment_size":200,
        "number_of_fragments":1,
        "fields":{
            "title":{

            },
            "provider":{

            },
            "description":{

            },
            "location":{

            }
        }
    }
}
```

Code snippet G.1: Query logging: Query 1

```
{
    "size":0,
    "query":{
        "bool":{
            "should":[
                {
                    "multi_match":{
                        "query":"sustainability",
                        "fields":[
                            "certificate_name^2.0",
                            "company_name^1.0",
                            "department^2.0",
                            "education_level^2.0",
                            "full_text^1.0",
                            "job_level^2.0",
                            "job_location^1.0",
                            "job_type^2.0",
                            "title^2.0"
                        ],
                        "type":"best_fields",
                        "operator":"OR",
                        "slop":0,
                        "prefix_length":0,
                        "max_expansions":50,
                        "zero_terms_query":"NONE",
                        "auto_generate_synonyms_phrase_query":true,
                        "fuzzy_transpositions":true,
                        "boost":1.0
                    }
                }
            ],
            "adjust_pure_negative":true,
            "boost":1.0
        }
```

```
    },
    "aggregations":{
        "salary_maximum":{
            "range":{
                "field":"salary_maximum",
                "ranges":[
                    {
                        "key":"0 - 1000",
                        "from":0.0,
                        "to":1000.0
                    },
                    {
                        "key":"1000 - 2000",
                        "from":1000.01,
                        "to":2000.0
                    },
                    {
                        "key":"2000 - 3000",
                        "from":2000.01,
                        "to":3000.0
                    },
                    {
                        "key":"3000 - 4000",
                        "from":3000.01,
                        "to":4000.0
                    },
                    {
                        "key":"4000 - 5000",
                        "from":4000.01,
                        "to":5000.0
                    },
                    {
                        "key":"5000 - 6000",
                        "from":5000.01,
                        "to":6000.0
                    },
                    {
                        "key":"6000 - 7000",
                        "from":6000.01,
                        "to":7000.0
                    },
                    {
                        "key":"7000+",
                        "from":7000.01
                    }
                ],
                "keyed":false
            }
        }
    },
    "highlight":{
        "fragment_size":200,
        "number_of_fragments":1,
        "fields":{
            "title":{

```

```
            },
            "provider":{

            },
            "description":{

            },
            "location":{

            }
        }
    }
}
```

Code snippet G.2: Query logging: Query 2

```
{
   "size":0,
   "query":{
      "bool":{
         "should":[
            {
               "multi_match":{
                  "query":"solar",
                  "fields":[
                     "certificate_name^2.0",
                     "company_name^1.0",
                     "department^2.0",
                     "education_level^2.0",
                     "full_text^1.0",
                     "job_level^2.0",
                     "job_location^1.0",
                     "job_type^2.0",
                     "title^2.0"
                  ],
                  "type":"best_fields",
                  "operator":"OR",
                  "slop":0,
                  "prefix_length":0,
                  "max_expansions":50,
                  "zero_terms_query":"NONE",
                  "auto_generate_synonyms_phrase_query":true,
                  "fuzzy_transpositions":true,
                  "boost":1.0
               }
            }
         ],
         "adjust_pure_negative":true,
         "boost":1.0
      }
   },
   "aggregations":{
      "salary_maximum":{
         "range":{
            "field":"salary_maximum",
            "ranges":[
               {
```

```
                    "key":"0 - 1000",
                    "from":0.0,
                    "to":1000.0
                },
                {
                    "key":"1000 - 2000",
                    "from":1000.01,
                    "to":2000.0
                },
                {
                    "key":"2000 - 3000",
                    "from":2000.01,
                    "to":3000.0
                },
                {
                    "key":"3000 - 4000",
                    "from":3000.01,
                    "to":4000.0
                },
                {
                    "key":"4000 - 5000",
                    "from":4000.01,
                    "to":5000.0
                },
                {
                    "key":"5000 - 6000",
                    "from":5000.01,
                    "to":6000.0
                },
                {
                    "key":"6000 - 7000",
                    "from":6000.01,
                    "to":7000.0
                },
                {
                    "key":"7000+",
                    "from":7000.01
                }
            ],
            "keyed":false
        }
    }
},
"highlight":{
    "fragment_size":200,
    "number_of_fragments":1,
    "fields":{
        "title":{

        },
        "provider":{

        },
        "description":{

        },
```

```
            "location":{

            }
        }
    }
}
```

Code snippet G.3: Query logging: Query 3

```
{
   "size":0,
   "query":{
      "bool":{
         "should":[
            {
               "multi_match":{
                  "query":"electrical",
                  "fields":[
                     "certificate_name^2.0",
                     "company_name^1.0",
                     "department^2.0",
                     "education_level^2.0",
                     "full_text^1.0",
                     "job_level^2.0",
                     "job_location^1.0",
                     "job_type^2.0",
                     "title^2.0"
                  ],
                  "type":"best_fields",
                  "operator":"OR",
                  "slop":0,
                  "prefix_length":0,
                  "max_expansions":50,
                  "zero_terms_query":"NONE",
                  "auto_generate_synonyms_phrase_query":true,
                  "fuzzy_transpositions":true,
                  "boost":1.0
               }
            }
         ],
         "adjust_pure_negative":true,
         "boost":1.0
      }
   },
   "aggregations":{
      "certificates":{
         "nested":{
            "path":"required_certificates"
         },
         "aggregations":{
            "names":{
               "terms":{
                  "field":"required_certificates.name",
                  "size":2147483647,
                  "min_doc_count":1,
                  "shard_min_doc_count":0,
                  "show_term_doc_count_error":false,
```

```
                    "order":[
                        {
                            "_count":"desc"
                        },
                        {
                            "_key":"asc"
                        }
                    ]
                }
            }
        }
    },
    "highlight":{
        "fragment_size":200,
        "number_of_fragments":1,
        "fields":{
            "title":{

            },
            "provider":{

            },
            "description":{

            },
            "location":{

            }
        }
    }
}
```

Code snippet G.4: Query logging: Query 4

```
{
    "size":0,
    "query":{
        "bool":{
            "should":[
                {
                    "multi_match":{
                        "query":"electrical",
                        "fields":[
                            "certificate_name^2.0",
                            "company_name^1.0",
                            "department^2.0",
                            "education_level^2.0",
                            "full_text^1.0",
                            "job_level^2.0",
                            "job_location^1.0",
                            "job_type^2.0",
                            "title^2.0"
                        ],
                        "type":"best_fields",
                        "operator":"OR",
                        "slop":0,
```

```
                    "prefix_length":0,
                    "max_expansions":50,
                    "zero_terms_query":"NONE",
                    "auto_generate_synonyms_phrase_query":true,
                    "fuzzy_transpositions":true,
                    "boost":1.0
                }
            }
        ],
        "adjust_pure_negative":true,
        "boost":1.0
    }
},
"aggregations":{
    "certificates":{
        "nested":{
            "path":"required_certificates"
        },
        "aggregations":{
            "names":{
                "terms":{
                    "field":"required_certificates.name",
                    "size":2147483647,
                    "min_doc_count":1,
                    "shard_min_doc_count":0,
                    "show_term_doc_count_error":false,
                    "order":[
                        {
                            "_count":"desc"
                        },
                        {
                            "_key":"asc"
                        }
                    ]
                }
            }
        }
    }
},
"highlight":{
    "fragment_size":200,
    "number_of_fragments":1,
    "fields":{
        "title":{

        },
        "provider":{

        },
        "description":{

        },
        "location":{

        }
    }
```

```
79      }
80  }
```

Code snippet G.5: Query logging: Query 5

Bibliography

[1] A. Mehlführer, Web scraping a tool evaluation, 2009. [Online]. Available: https://www.big.tuwien.ac.at/app/uploads/2016/10/Mehlf%5C%C3%5C%BChrer_paper.pdf.

[2] Beautiful Soup developers, Beautiful soup documentation, 2021. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (visited on 01/19/2021).

[3] Scrapy developers, Scrapy 2.4 documentation, 2021. [Online]. Available: https://docs.scrapy.org/en/latest/ (visited on 01/19/2021).

[4] Cheerio developers, Cheerio, 2021. [Online]. Available: https://cheerio.js.org/ (visited on 01/19/2021).

[5] GNOME project, Reference manual for libxml2, 2020. [Online]. Available: http://www.xmlsoft.org/html/index.html (visited on 11/16/2020).

[6] Beautiful Soup developers, Installing a parser, 2020. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser (visited on 11/13/2020).

[7] Xml and html with python. [Online]. Available: https://lxml.de/.

[8] Overview. [Online]. Available: https://html5lib.readthedocs.io/en/latest/.

[9] Cheerio developers, Cheerio, 2021. [Online]. Available: https://github.com/cheeriojs/cheerio (visited on 01/21/2021).

[10] M. Fowler, Inversion of control, 2021. [Online]. Available: https://www.martinfowler.com/bliki/InversionOfControl.html (visited on 01/21/2021).

[11] G. Van Rossum et al., "Python programming language.," in USENIX annual technical conference, vol. 41, 2007, p. 36.

[12] B. W. Kernighan and D. M. Ritchie, The C programming language. 2006.

[13] C. Severance, "Javascript: Designing a language in 10 days," Computer, vol. 45, no. 2, pp. 7–8, 2012.

[14] D. Frampton, S. M. Blackburn, P. Cheng, R. J. Garner, D. Grove, J. E. B. Moss, and S. I. Salishev, "Demystifying magic: High-level low-level programming," in Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2009, pp. 81–90.

[15] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[16] Xapian java bindings. [Online]. Available: https://xapian.org/docs/bindings/java/.

[17] Elasticsearch reference. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html.

[18] Apache solr reference guide: Apache solr reference guide 8.7. [Online]. Available: https://lucene.apache.org/solr/guide/8_7/.

[19] Documentation. [Online]. Available: https://manual.manticoresearch.com/Introduction.

[20] Documentation. [Online]. Available: https://xapian.org/docs/.

[21] Elasticsearch vs manticore search: What are the differences? [Online]. Available: https://stackshare.io/stackups/elasticsearch-vs-manticore-search.

[22] TechTarget Contributors, What is rest api (restful api)? Sep. 2020. [Online]. Available: https://searchapparchitecture.techtarget.com/definition/RESTful-API#:~:text=A%20RESTful%20API%20is%20an,deleting%20of%20operations%20concerning%20resources..

[23] M. Awad, "A comparison between agile and traditional software development methodologies," Ph.D. dissertation, 2005.

[24] C. Drumond, Scrum - what it is, how it works, and why it's awesome. [Online]. Available: https://www.atlassian.com/agile/scrum.

[25] M. Lotz, Waterfall vs. agile: Which methodology is right for your project? Nov. 2018. [Online]. Available: https://www.seguetech.com/waterfall-vs-agile-methodology/ (visited on 11/12/2020).

[26] Your tool for style guide enforcementű. [Online]. Available: https://flake8.pycqa.org/en/latest/.

[27] Pluggable javascript linter. [Online]. Available: https://eslint.org/.

[28] A. Amanse, Why should you use microservices and containers? [Online]. Available: https://developer.ibm.com/depmodels/microservices/articles/why-should-we-use-microservices-and-containers/.

[29] P. A. Castillo, P. Garca-Sánchez, M. G. Arenas, A. M. Mora, G. Romero, and J. J. Merelo, "Using soap and rest web services as communication protocol for distributed evolutionary computation," International Journal of Computers & Technology, vol. 10, no. 6, pp. 1752–1770, 2013.

[30] Scrapy developers, Selecting dynamically-loaded content, 2020. [Online]. Available: https://docs.scrapy.org/en/latest/topics/dynamic-content.html (visited on 11/16/2020).

[31] M. Lindsaar, Postgresql vs mysql, Oct. 2018. [Online]. Available: https://medium.com/@articles_92466/postgresql-vs-mysql-fe9d65887520.

[32] K. Hristozov, Mysql vs postgresql, Jul. 2019. [Online]. Available: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres#:~:text=Postgres%20is%20an%20object-relational,more%20closely%20to%20SQL%20standards..

[33] Useformik(). [Online]. Available: https://formik.org/docs/api/useFormik.

[34] Jquense, Jquense/yup. [Online]. Available: https://github.com/jquense/yup.

[35] Alpine linux. [Online]. Available: https://distrowatch.com/table.php?distribution=alpine.

[36] T. Bui, "Analysis of docker security," arxiv.org, [Online]. Available: https://arxiv.org/abs/1501.02967.

[37] W. Croft, D. Metzler, and T. Strohman, "Evaluating search engines," in Search Engines, Information Retrieval in Practice. Pearson Education, Inc., 2015, ch. 8.

[38] Y. Xu and Z. Chen, "Relevance judgment: What do information users consider beyond topicality?" Journal of the American Society for Information Science and Technology, vol. 57, no. 7, pp. 961–973, 2005.

[39] J. A. Aslam, V. Pavlu, and R. Savell, "A unified model for metasearch, pooling, and system evaluation," in Proceedings of the Twelfth International Conference on Information and Knowledge Management, ser. CIKM '03, New Orleans, LA, USA: Association for Computing Machinery, 2003, pp. 484–491, ISBN: 1581137230. DOI: 10.1145/956863.956953. [Online]. Available: https://doi.org/10.1145/956863.956953.

[40] Y. Lv and C. Zhai, "Lower-bounding term frequency normalization," ser. CIKM '11, Glasgow, Scotland, UK: Association for Computing Machinery, 2011, pp. 7–16, ISBN: 9781450307178. DOI: 10.1145/2063576.2063584. [Online]. Available: https://doi.org/10.1145/2063576.2063584.

[41] Similarity module, 2021. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html (visited on 01/20/2021).

[42] A. Hayes and K. Krippendorff, "Answering the call for a standard reliability measure for coding data," Communication Methods and Measures, vol. 1, pp. 77–89, Apr. 2007. DOI: 10.1080/19312450709336664.

[43] K. Krippendorff, "Computing krippendorff's alpha-reliability," Jan. 2011.

[44] C. Van Gysel and M. de Rijke, "Pytrec_eval: An Extremely Fast Python Interface to trec_eval," arXiv e-prints, arXiv:1805.01597, May 2018. arXiv: 1805.01597 [cs.IR].

[45] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," Commun. ACM, vol. 32, no. 12, pp. 1415–1425, Dec. 1989, ISSN: 0001-0782. DOI: 10.1145/76380.76382. [Online]. Available: https://doi.org/10.1145/76380.76382.

[46] De Banenmakers, Rotterdam Werkt, 2021. [Online]. Available: https://debanenmakers.nl/rotterdam-werkt (visited on 01/18/2021).