BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

DATA MINING BASED WEB ANALYZER OF JOB ADVERTISEMENTS

WEBOVÝ ANALYZÁTOR PRACOVNÍCH INZERÁTŮ S VYUŽITÍM DATA MININGU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Alex Wittner

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. Marek Sikora

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Bachelor's Thesis

Bachelor's study program **Information Security**

Department of Telecommunications

Student: Alex Wittner ***ID:*** 230914

Year of study: 3 ***Academic year:*** 2022/23

TITLE OF THESIS:

Data Mining Based Web Analyzer of Job Advertisements

INSTRUCTION:

The goal of this thesis is to extend functionalities of the existing web application for the analysis of job advertisements. The application contains a form for inserting job advertisements into the internal database (the sources are advertisements on various web portals, e.g. LinkedIn). The student will create automation for adding new advertisements to the database. The automation will work as follows. The administrator will want to add the ad to the internal database. In the form, he enters a hyperlink to a web ad, the automation analyzes its content and, based on that, pre-fills the form to add the ad to the internal database.

The first task of this thesis is to create a data mining method to search for keywords, set the parameters of the job and test the accuracy on an existing database. Next task is the optimization of the search accuracy, completing the missing data in the database, and integrating the final tool into the web application.

RECOMMENDED LITERATURE:

Podle pokynů vedoucího práce.

Date of project specification: 6.2.2023 ***Deadline for submission:*** 26.5.2023

Supervisor: Ing. Marek Sikora

doc. Ing. Jan Hajný, Ph.D.
Chair of study program board

ABSTRACT

The goal of this bachelor's thesis was to create an automated submission of new job postings by inserting a URL within an existing web application (`https://rewire.informacni-bezpecnost.cz`) that aims to collect job postings in the cybersecurity field with a detailed job competency analysis. The job advertisements are analyzed using a multi-pattern search algorithm, Aho-Corasick, written in Java. A Python script using the Selenium library extracts information from job advertisements. The resulting implementation and web page are created using PHP and the ReactJS library using JavaScript.

KEYWORDS

Web Application, Job Ads Analyzer, ReactJS, PHP, Java, Python, Selenium, cybersecurity, multi pattern search algorithm, Aho-Corasick, Commentz-Walter, dictionary, job ads, Rewire, web scraping, data mining

ABSTRAKT

Cílem této bakalářské práce bylo vytvoření automatizovaného zadávání nových pracovních inzerátů pomocí vložení URL v rámci již existující webové aplikace (`https://rewire.informacni-bezpecnost.cz`), jejíž cílem je shromažďování pracovních inzerátů v oblasti cybersecurity s podrobnou analýzou pracovních kompetencí. Pracovní inzeráty jsou analyzovány pomocí více vzorového vyhledávacího algoritmu Aho-Corasick, psaného v jazyce Java. K získávání informací ze zadaných pracovních inzerátů slouží Python skript využívající knihovnu Selenium. Výsledná implementace a webová stránka je vytvořena pomocí jazyka PHP a knihovny ReactJS využívající JavaScript.

KLÍČOVÁ SLOVA

Webová aplikace, Analyzátor pracovních inzerátů, ReactJS, PHP, Java, Python, Selenium, cybersecurity, více vzorový vyhledávací algoritmus, Aho-Corasick, Commentz-Walter, slovník, pracovní inzeráty, Rewire

ROZŠÍŘENÝ ABSTRAKT

Cílem této bakalářské práce bylo vytvoření automatizovaného zadávání nových pracovních inzerátů pomocí vložení URL v rámci již existující webové aplikace (`https://rewire.informacni-bezpecnost.cz`), jejíž cílem je shromažďování pracovních inzerátů v oblasti cybersecurity s podrobnou analýzou pracovních kompetencí. Pracovní inzeráty jsou analyzovány pomocí více vzorového vyhledávacího algoritmu Aho-Corasick, psaného v jazyce Java. K získávání informací ze zadaných pracovních inzerátů slouží Python skript využívající knihovnu Selenium. Výsledná implementace a webová stránka je vytvořena pomocí jazyka PHP a knihovny ReactJS využívající JavaScript.

Motivace vzniku práce

Motivací k vytvoření tohoto projektu byl dlouhodobý problém s nedostatkem kvalifikovaných pracovníků v oblasti IT/bezpečnosti. Odhady počtu chybějících pracovníků se v Česku pohybují mezi 20 až 30 tisíci. Chybí také platforma/aplikace, která by komplexně řešila a shromažďovala pracovní inzeráty z různých zdrojů a zobrazovala požadované dovednosti pro uchazeče o zaměstnání více konkrétněji a přehledněji.

Webová aplikace poskytuje tuto možnost s mnoha důležitými funkcemi, jako jsou např. filtrování podle místa nabídky, webu na kterém je inzerát umístěn, a jiné. Webová aplikace obsahovala funkci pro vkládání nových pracovních inzerátů, ale pouze pomocí ručního zadání. Všechny položky tudíž musí zadat uživatel, včetně požadovaných dovedností, které musí vyčíst z popisu pracovního inzerátu. Aby uživatel nemusel složitě vyhledávat v textu požadované dovedností a ručně vyplňovat údaje z inzerátu, je možné použít automatizace sestávající z webového "scraperu" napsaného v jazyce Python a hlavní části bakalářské práce textového analyzátoru, napsaného v jazyce Java. Jeho úkolem bude pomocí pokročilého algoritmu Aho-Corasick vyhledávat v textu slova, která jsou součástí slovníku. Vytvoření slovníku je také úkolem bakalářské práce.

Teoretická část

Teoretická část se dělí na 2 části.

V první části se čtenář dozví více infromací o webech, webových aplikacích, rozdílech mezi nimi a ostatních webových nástrojích. V rámci popisu webových aplikací dojde také k ukázání výhod a nevýhod jejich použití. Zmíněn je také krátký odstavec pojednávající o historii webů a podrobnějších detailech o použitých webových technologiích. V rámci webových technologií dojde k detailnějšímu seznámení jak tyto technologie fungují, jaké jsou jejich výhody a nevýhody a výtažek

z jejich historie. V rámci technologií budou popsány jak statické, tak dynamické webové stránky a aplikace, stejně tak zda fungují spíše na straně klienta, serveru, či mohou běžet na obou stranách zároveň.

Další podkapitola se zaměřuje na použité programovací jazyky netýkající se webového vývoje. Konkrétně popis fungování jazyků Java, Python a v prostředí Python navíc popis knihovny Selenium. Součástí kapitol bude zdůraznění předností jednotlivých jazyků společně s jejich výhody, nevýhody, použití a případné porovnání s ostatními vývojovými jazyky. U knihovny Selenium použité v rámci Python skriptu dojde k představení čeho je knihovná schopná, k čemu slouží a jak se využívá v praxi.

Druhá část popisuje stěžejní bod této práce a tím je zpracování dat. Dojde k objasnění termínů jako: "data mining, data scraping, pattern searching", vysvětlení jednotlivých algoritmů pro vyhledávání a proč v rámci této práce byl nejlepším vhodným algoritmem, takový algoritmus, jenž se řadí do skupiny více vzorových vyhledávacích algoritmů. V rámci více vzorových vyhledávacích algoritmů dojde k detailnímu popisu fungování algoritmů Aho-Corasick a Commentz-Walter.

Implementace a Shrnutí

V další kapitole Implementace je vysvětleno a popsáno jaké kroky byly provedeny pro vytvoření programů, skriptu a jejich zprovoznění v rámci webové aplikace také jak vznikaly slovníky potřebné pro více vzorový vyhledávací algoritmus. Je možné zde nalézt ukázky kódu, jenž byly použity a jaké problémy se při tvorbě této práce objevili a jak byly vyřešeny.

V části Shrnutí dojde ke shrnutí dosažených výsledků, primárně pak vysvětlení, který z navrhnutých více vzorových vyhledávacích algoritmů je lepší a také proč a jak proběhlo jejich porovnání, když se druhý algoritmus implementovat nepodařilo, tak jak bylo původně zamýšleno. Stejně tak dojde k objasnění, který ze slovníků byl použit a proč.

Závěr

V závěru jsou ustáleny dosažené výsledky a předpokládané další kroky na tomto projektu.

WITTNER, Alex. *Dictionary search algorithm for job ads profiling*. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2023, 81 p. Bachelor's Thesis. Advised by Ing. Marek Sikora

Author's Declaration

Author:	Alex Wittner
Author's ID:	230914
Paper type:	Bachelor's Thesis
Academic year:	2022/23
Topic:	Dictionary search algorithm for job ads profiling

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the paper and listed in the comprehensive bibliography at the end of the paper.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to thank my thesis supervisor Ing. Marek Sikora and the thesis consultant M.Sc. Sara Ricci Ph.D., for professional guidance, consultation, patience, and inspiring suggestions for the thesis.

Contents

List of Figures

Listings

Introduction

The motivation to create this project was the long-lasting problem of the need for more qualified workers in the IT/security sphere. Estimates of the number of missing workers are between 20 and 30 thousand in Czechia [1]. There is also a lack of frameworks on the Internet that would comprehensively handle the job of advertisements from different sources and display the required skills for candidates more precisely and clearly.

The web application provides this option with many essential features, such as filtering by the location of the job, the site on which the ad is placed etc. The web application currently includes a function for inserting new job adverts, but for this moment, it is only possible to insert job adverts by manual input. i.e. all items must be entered by the user, including the required skills, which must be read from the job description. It is the job description that can give us information about the required job skills that are not directly stated in the job requirements.

To prevent the user from having to search complexly in the text for the required skills and manually fill in the data of the job advert, an automation consisting of a web scraper written in Python and the main part of the bachelor thesis, a text analyser written in Java was created. It will use advanced algorithms to search the text for words that are part of the dictionary. The creation of the dictionary is also the task of the bachelor thesis. Different ways will help to create it, for example, individual definitions of the required skills according to NIST (*National Institute of Standards and Technology*).

The Python script using Selenium will be used for web scraping, which is a tool for automated work with web applications.

The algorithms considered in this thesis will be Aho-Corasick and Commentz-Walter. The first algorithm Aho-Corasick is one of the best dictionary search algorithms. Thanks to working with the trie data structure, which can be thought of as a sort of binary tree for characters. Another factor affecting its efficiency is that the algorithm works with all the patterns from the dictionary at once, as opposed to slower algorithms such as Knuth-Morris-Pratt (*KMP*), which only works with one pattern at a time.

The second algorithm is the Commentz-Walter algorithm, which is also a perfect searching algorithm which consists of two very efficient algorithms, the Aho-Corasick mentioned above algorithm and the Boyer-More algorithm, which is used for string search. Commentz-Walter also works on the trie and multiple pattern search principle, complemented by indexing.

The result of the Java program using one of the algorithms will be the words that match the given string with the dictionary and the number of words in each skill group category to which they belong. This subsequent data will be used and displayed on the web.

1 Overview of Used Technologies

In this chapter, the reader will be introduced in more detail to what web and web applications are and the tools for creating web and web applications used in their development.

1.1 World Wide Web

WWW (*World Wide Web*) is a system of public web pages and websites interconnected through the Internet. A web page is a file or a group of files (typically hypertext documents) that allows the user to view online content. It is mainly displayed through the World Wide Web. For the user to be able to view this content, the user needs to have access to the Internet and a web browser. Files need to be stored somewhere. This place is called a web server. A web server delivers web content to the user when the user makes a request through HTTP (*Hypertext Transfer Protocol*). Usually, it is provided by entering a domain name into the URL (*Uniform Resource Locator*) address in the web browser. This allows the user to see the requested web page. A website is a collection of many web pages associated under one domain name. [2]

History of World Wide Web

World Wide Web (*WWW*) was created in 1989 by British computer scientist Tim Berners-Lee at CERN (*European Organization for Nuclear Research*). Initially, this system was created as a document management system. At the end of 1990, the function system was already implemented with the HTTP server and WWW browser. A couple of months later, in January 1991, the WWW was provided to the other research institutions by CERN and in the summer of 1991, it was provided for general public usage. After CERN made the World Wide Web free to the general public in 1993, the WWW started to take off in a big way.

World Wide Web Consortium (*W3C*), which was created by Tim Berners-Lee, made it possible to create XML (*Extensible Markup Language*). Tim Berners-Lee proposed to replace the original HTML with stricter XHTML (*Extensible Hypertext Markup Language*) in 1996. However, thanks to the fact that other developers managed to find out the principle of XMLHttpsRequest functioning, the web revolution called Web2.0 started. Mozilla, Opera and Apple formed a community of people (WHATWG) to develop HTML5 because they refused to use XHTML. In 2019, XHTML was abandoned and the W3C group started to participate in the WHATWG. [4]

1.2 Web Application

A web application is a software application which has the unique feature that, unlike most software applications, it does not need to be installed on the user's device. This is possible due to the fact that the internet browser handles this software. Web applications are visually the same as ordinary web pages. They even work on the same basis and languages, namely HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*), JavaScript, PHP (*Hypertext Preprocessor, originally Personal Home Page*), SQL (*Structured Query Language*), etc. However, the difference compared to web pages is that they have many more functions and options. The difference can be seen to a lesser extent in their use. It is not necessarily the rule and considering how new technologies and possibilities of both pages and web applications are changing. It is possible that in the future, these differences will partially disappear. Currently, the differences are the following: Web pages nowadays serve as a preferred way of presenting companies, individuals, a certain topic or a range of topics, or anything that someone finds interesting or important. Web applications are used in the form of interactivity/diversity. Their purpose is not only to share something with the user but also to work with them and provide them with possible services and possibilities. Among this, it is possible to imagine, for example, the use of an online form, a basket for online shopping, or the actual work with office packages in an online form (Google Docs, Google Sheets, etc.). Web applications can therefore be considered in part as computer programs. [2] [3]

1.2.1 Advantages and Disadvantages of Web Applications

Advantages:

- Removing compatibility issues because all users access the same version.
- Lower requirements for the end user's computer.
- Easier and more reliable security for paid sites from a piracy perspective.

Disadvantages:

- Worse optimization for SEO (*Search engine optimization*) [2].

1.3 Web Development Technologies

Various technologies are used for web development. The most common are HTML, CSS, PHP, SQL and JavaScript. These will be discussed more in the following subsections. In addition, the React technology used for the bachelor thesis web application, which is one of the most advanced technologies in use nowadays, will also be described.

1.3.1 HTML

HTML is a hypertext markup language designed to create simple web pages. HTML was created in 1993. It's the basic building block of most web pages or web applications. Since its creation, it has, of course, undergone many modifications and improvements. Today it used a version called HTML5. Individual versions will be discussed later. The original version had three primary purposes [5].

1. Creating,
2. formatting,
3. styling web pages.

Styling is no longer used because this feature has been replaced by cascading styles, which have many more options for the visual editing of pages. Styling is still possible within HTML but is not recommended or welcomed.

HTML uses so-called tags for its work. They are parts of the code located mostly between < and > characters. Tags can be sorted into paired and unpaired. Unpaired tags contain a single command between <> characters. Paired tags are marked by starting as <command> and ending with the same command name with the / character added, so the ending pair tag looks like </command>. Tags can be combined, nested, and aligned with each other, which, together with cascading styles, makes it possible to display practically anything. The code structure is partly up to everyone, but when creating an HTML document, it is necessary to respect the basic syntax and logic. The international W3C consortium maintains these rules. Each document must contain information about the document type, the root element html, a header that contains the formal content of the document and the content which is not the content of the page (not displayed on the page), then the title that is in the header tag. The most comprehensive part of the code is the body of the page itself, which contains the content that will be displayed to the user. Tags that must have every HTML document are:!DOCTYPE html, html, head, title, and body [5].

Also worth mentioning are the so-called WYSIWYG (*What You See Is What You Get*) editors, which allow users who do not know HTML syntax to create web pages. As the name suggests, the only task of the web designer is selecting, placing and overwriting the objects that the user has in the menu. This is possible thanks to huge libraries with already created objects. The content of the code is fully automatically created in the background by the selected editor. [5] [6]

HTML Versions

A brief overview of the most important versions that have been introduced for HTML over the years:

- **Version 0.9 – 1.2** – not supporting GUI, created by Tim Berners-Lee and Daniel Connolly, 1991-1993.
- **Version 2.0** – graphics support, adds interactive formulas and entirely depends on SGML (standard generalized markup language) syntax, released by IEFT (Internet Engineering Task Force) community, 1995.
- **Version 3.2** – added tables, stylesheets, text-alignment, simplified from version 3.0, which was too complex, added by W3C community, January 1997.
- **Version 4.0** – standardization of frames added, new elements added for tables and forms, end of support for little used elements, some existing elements modified, use of cascading styles for styling, W3C, December 1997.
- **Version 4.01** – last version using HTML before switching to XHTML, bug fixes for the previous version, W3C, December 1999.
- **Version 5.0** – a lot of changes, the most significant include the addition of new semantic elements, support for applications running without an Internet connection, support for multimedia directly in the browser (video, audio, etc.), fixing broken elements, large cleanup of unused tags, simplification of used elements especially in terms of writing and use, W3C, October 2014 [7].
- **Version 5.2** – currently used version, fixed, added new features, simplified and less error-prone Payment Request API, new features for better security, W3C, December 2017 [8]. [9]

1.3.2 CSS

CSS is a stylesheet language used to style pages written in HTML, XHTML and XML markup languages. Like HTML, it can be considered one of the basic building blocks of web pages and applications. The reason for its creation was straightforward. Before CSS styles started to be used, editing visual elements of websites took a lot of work and effort. Each element had to contain parameters of its appearance, which made the code much longer and less clear. Another big disadvantage of HTML styling was that if the creator wanted to change certain appearance groups for all tags (for example, h1 headings), they had to access each such tag individually and change the parameter for each. Therefore, CSS was created, which offered a simple alternative instead of long and complex styling. [10]

History and Versions of the CSS

In 1996, the first version of CSS, CSS1, was written by Håkon Wium Lie. Three months later, in February 1997, a separate group was formed in the W3C to work on and maintain CSS as well as HTML. CSS2 was released by the W3C in November 1997, with features that CSS1 did not offer (the most important new features could include outline parameters, max, min-height and weight, position, visibility, etc.). The official recommendation for using CSS 2.1 came out in 2011 after many modifications that were gradually taking place. Currently, a version of CSS without version marking is used. There were also CSS3 and CSS4 versions in development. The CSS3 version was expected mainly because of the new features associated with the latest version of HTML, HTML5. However, after version 2.1, its developers (*W3C*) decided that instead of releasing new versions, they would periodically create snapshots describing the new modules that have been added for the reason of easier language management. [11]

Advantages and Disadvantages of the CSS

Advantages:

- More comprehensive formatting options,
- faster page loading,
- faster and easier styling and changing styling within the whole document,
- possibility of advanced modifications in cooperation with JavaScript, for example, changing the parameters on the runtime.

Disadvantages:

- Different browsers may display different content. [10]

1.3.3 PHP

PHP is an open-source scripting programming language. It creates mostly dynamic web pages, web applications and other systems. It can also be used for creating desktop applications or command line scripting (e.g. for writing cron or Task Scheduler).

The PHP language can be embedded in HTML, which in some cases, greatly simplifies the code and opens up many new possibilities offered by the PHP language functions. The reason for using it is straightforward, PHP provides more features in creating web pages and web apps and sometimes faster page loading than static pages. That is possible because the code is executed on the server side, and the user receives the finished output. The user does not need to perform any installation and does not know the difference in terms of availability. [12] [13]

Another advantage of the PHP language is that it is cross-platform, meaning it can be run on any major operating system without compatibility problems. It is considered one of the simplest languages in terms of syntax and learning. It has a wide range of applications and, together with SQL, works very well with most of the well-known database systems (e.g. MySQL, PostgreSQL, MSSQL and many others).

Despite the fact that PHP seems to be on the way out, this is not necessarily the case. It is still a widely used language in the world, thanks to which some of the most famous websites such as Wikipedia, WordPress and Facebook have been created and run. According to the PYPL(Popularity of Programming Language) website, it is ranked 6th among used languages. Also, on the TIOBE website, according to the popularity index, it is in the 10th position. The positions are current as of 15 November 2022. [12] [13]

1.3.4 SQL

SQL is a standardized structured query language used to create, manage and manipulate relational databases and the associated data. Like a classic database, a relational database stores a large amount of data with a fixed record structure. In the case of relational databases, the data is stored as tables with rows and columns representing different attributes. The main advantage of the SQL language is that it can be used very efficiently within other programming languages. The SQL language is very simple and intuitive. Rather than a complex programming/query language, it is a kind of "machine" English, more complex questions do not have to be so simple. That is not due to the language itself but rather to the logical complexity of the query.

Well-known database systems include, for example, MySQL, PostgreSQL, Oracle and Microsoft SQL Server. All these servers work with SQL. Different advanced commands and properties may vary from system to system, but basic SQL commands such as SELECT, CREATE, INSERT, UPDATE, DELETE, DROP, and many others are the same in most if not all the systems [14].

Sorting According to Use

The SQL language commands can be sorted according to their usage as follows:

- **Data definition language (DDL)** – Commands belonging to this group allow to create, edit and delete objects in the database, for example, CREATE.
- **Data manipulation language (DML)** – Commands belonging to this group allow to create, edit and delete data in the database, for example, INSERT.

- **Data query language (DQL)** – Commands belonging to this group can retrieve data stored in the database, for example, SELECT.
- **Data control language (DCL)** – Commands belonging to this group allow you to manage access permissions for users to the database, for example, GRANT.
- **Transaction control language (TCL)** – Commands belonging to this group allow to manage transactions in the database, for example, COMMIT, ROLLBACK.

1.3.5 JavaScript

JavaScript is a scripting-programming language that allows the creation of dynamic web pages and applications. It is mainly used on the client side. Thanks to technologies like Node.js and similar, it is possible to use JavaScript on the server side, just like PHP. However, it is a kind of JavaScript superstructure, so JavaScript itself is used more on the client side.

Parts of JavaScript code are called scripts; they should be part of the HTML code or refer to it. Adding these scripts to an HTML document opens up new possibilities for interaction with the user. These interactions can be new page animations, text visibility adjustments, and more modern and nice-looking drop-down menus. Pages without JavaScript have these capabilities, but only on the level of static web pages, which are not so diverse. Unlike PHP, JavaScript does not burden the server so much. While PHP downloads and processes the whole page on the server, JavaScript, which works on the client side, only sends a request to the server. The server sends back HTML and script, which is then processed by the browser and displayed to the user. [15]

Advantages and Disadvantages of JavaScript

Advantages:

- Speed of loading – The user does not have to wait for the page to be reloaded because it runs on their side. In case of an action, they do not have to reload the whole page again, but only the changed part.
- Less server load – The request is checked on the client side (web browser) before sending it. This point is also related to the loading speed point.
- Greater possibilities of interaction – Contains a lot of libraries dealing with new functions, appearance and properties of pages and applications, and dynamic structures.
- Extensibility – Lots of frameworks, simplifying application development.
- Compatibility – Can work with different languages like PHP, Java and others.

Disadvantages:

- Browser compatibility – each web browser and its version can handle JavaScript differently so that the resulting displays may differ, or JavaScript may be disabled completely.
- Security – scripts can be displayed as part of HTML, which can be a potential danger for code abuse. [16]

1.3.6 ReactJS

React, or React.js, is an open-source JavaScript front-end library developed by Facebook. It is used to create web applications and web user interfaces. React allows you to build websites quickly with more options and possibly shorter code than JavaScript alone. Thanks to this, React is one of the most popular and most used JavaScript libraries of the present. The fact that React is one of the best choices for developing web applications and interfaces are proven by the fact that it is used by some of the most famous companies in the world, such as Facebook, Instagram, Netflix, Reddit, Uber, Airbnb and thousands of others [17].

The first mentions of React date back to 2011, when Facebook started developing it for its own needs. In 2013 it was released by Facebook after previous testing and use within its own platform. The emergence of React had a massive benefit because of how creatively its approach and work with DOM (*Document Object Model*) were handled.

DOM is an interface that turns an HTML or XML document into a tree-based, logical structure, where each node represents a part of the document. In memory, these parts are called objects [18].

React allows developers to interfere with the DOM without needing knowledge of DOM operations. That is achieved thanks to the virtual DOM, which React itself creates based on the programmed code, the content of which is mainly how the resulting document should look. It then compares the virtual DOM with the real DOM and, thanks to clever algorithms, modifies only the parts of the real DOM that need to be changed in case of the differences. [19].

Benefits of using React

- Simple syntax, where it is possible to use HTML tags.
- Extensive possibilities to create user environments.
- Large number of extensible libraries can be used to enliven applications.
- Faster rendering thanks to the use of virtual DOM.
- All visible components can display their current data.

- Allows decomposition into components, which can be used to split the code into smaller parts. That makes the code clearer and easier to manage.
- Possibility to create mobile applications.

React and Model View Controller (MVC) Architecture

Model View Controller is an architecture used mainly in web application development. Its function is to divide the application into three components, model, view and controller. Thanks to this, it is possible to work with the given components independently. React works only with the view component, which, as the name suggests, determines how the data will be displayed and represented in the browser.

1.4 Other Development Technologies

This part will focus on describing two programming languages: Python and Java. It will be shortly described how the languages work, what they can do and where it is appropriate to use them and also their advantages and disadvantages.

1.4.1 Java

Java is a multiplatform object-oriented programming language created in 1995 by Sun Microsystems. It is used for developing desktop, web and mobile applications.

Among the most common uses of Java are: game development, cloud computing, big data analysis, artificial intelligence, and the internet of things (IoT) [20].

The great advantage of Java is that it is possible to use Java not only in any operating system but also in hardware architectures without complex modifications. That is possible thanks to the Java Virtual Machine (JVM), which is a kind of intermediary between the code and the used system.

Java works on the principle of compilation, which means that the code written using the Java language is then compiled into bytecode, which is further compiled using the JVM into machine code that is required on the device/system where the program runs. [21]

Advantages and Disadvantages of Java

Advantages:

- Platform Independent – Java uses a Java Virtual Machine, which can be run almost everywhere.

- Secured – Java doesn't use Explicit pointers and runs inside the virtual machine sandbox, separating class packages from the local file system from those imported through the network.
- Memory Management – Java has a garbage collector that automatically frees up memory that is no longer in use, which helps prevent memory leaks and other memory-related issues.
- Multi-Threaded – Java uses a multi-threaded environment where a larger job can be divided into many threads and performed independently. The major advantage of multi-threading is that it does not have to allocate memory to each operating thread.

Disadvantages:

- Performance – Slower performance compared to lower-level languages like C and C++.
- Memory Consumption – Java uses more memory because it runs on a Java Virtual Machine.
- Poor GUI – GUI builders developed within the framework of Java does not allow such a variety and possibility of creating complex UIs, as is the case with other programming languages such as C# or Python. [22]

1.4.2 Python

Python is an interpreted, object-oriented, high-level programming language developed by Guido van Rossum in 1991. Interpreted means that the source code is executed line by line by the interpreter, not compiled into machine code. When you run a Python program, the interpreter reads the code and executes it directly without needing a separate compilation step, but this is not always true. For the most part, Python is an interpreted language, not a compiled language, although compilation is one of the steps. Python code written in a .py file is first compiled into byte code and stored in .pyc or .pyo format.

In terms of high-level functionality, its built-in data structures, together with dynamic typing and dynamic binding, make it very attractive for quicker development as well as usage as a scripting or glue language to connect existing components. [23] [24]

Python is used for various applications, for example, web development, scientific computing, data analysis, artificial intelligence, machine learning, automation or scripting and many more.

Advantages and Disadvantages of Python

Advantages:

- Easy to Learn – Python has a simple syntax that is easy to read and write.
- Large Library – Python has a large and comprehensive standard library that includes modules for a wide range of tasks.
- Multiplatform – Python code can run on multiple platforms like Windows, Linux, and macOS.

Disadvantages:

- Performance – Because Python is an interpreted language, it can be slower than compiled languages like C++.
- Mobile app development – Python is not suitable for mobile app development as it can be slow and has limited support for mobile app development.
- Memory consumption – Python programming language uses a large amount of memory.

Selenium

Selenium is an open-source tool used to automate web browsers. It allows creation automation that can simulate real user interactions with websites and web applications, reading content, clicking buttons, filling out forms and others.

Selenium can be used, for example, for scraping web pages. Together with the Python language, it is possible to create scripts that automate web scraping tasks and then use the collected data for various purposes, such as data analysis, machine learning or creating web applications.

Selenium can also be used for a variety of other tasks, such as automating web testing, generating screenshots of web pages, and performing stress tests on web applications. [25]

2 Data Processing

This theory section will discuss the data processes used in this thesis, specifically data mining, pattern searching and web scraping, in more detail.

2.1 Data Mining

Data mining is an analytical process during which a large, sometimes even opaque, amount of data is processed to obtain valuable data. Initially, it was used mainly in commercial areas, the use of data mining was different, but the goal was the same: to increase profits and related issues to them. In scientific research, it was mainly used to analyze information related to genetics. Over time, it has found use and scalability for various industries and is no longer seen as a tool primarily for commercial use. After its generalization, data mining can be divided into 6 phases.

1. Determining the goal – Right at the beginning, it should be thought about the goal of data mining, why to use it, the expected output, and how it will be handled.
2. Understanding the data – In this step, it is necessary to understand what the input data of the process will look like, how it should be handled and which data can be useful and of quality and which not.
3. Data preparation – Determining where the data will be collected from, its subsequent editing, cleaning, and resolving incorrect inputs so that the data can be well handled and analyzed.
4. Model building – Choose what kind of data mining will be used based on the expected results, create the model, edit, and test it.
5. Evaluation of results – Evaluate the results, whether the data obtained are useful, and test that they are reliable and correct. Part of the evaluation should be whether to implement the process or not.
6. Implementation and monitoring – When a process is approved, it is implemented, and further steps are taken based on the information found. Over time, it is necessary to check that the data collected is still up-to-date and that the models are maintained.

For data mining, many different techniques and methods are used, which allow to extraction of useful data from a large amount of data. It is certainly not possible to list them all, but a few of them are worth mentioning: Decision trees, association rules, neural sites, k-nearest neighbour (KNN), and clustering. [26] [27]

2.1.1 Web Scraping

Web scraping is a part of data mining because it is a technique of collecting data from the web in order to perform data mining analysis.

Web scraping is the process of extracting data from web or web applications. Software tools automatically extract data from websites and web applications according to predefined parameters to obtain information. Nowadays, many sites are aware of the use of these techniques, and to avoid server overload, they use various restriction methods on their sites. One of the simplest scraping methods is the cURL library, which is part of almost every programming language. cURL allows downloading any part of the HTML code on the requested web page for further downloading. A major disadvantage of cURL is the inability to work with sites whose content is dynamic (for example, written in JavaScript).

Another tool used for web scraping is the Selenium library, which allows access and retrieval of information from dynamic pages. Its disadvantage, however, is the need for up-to-date mapping.

2.2 Algorithms Used for Pattern Search

As the name suggests, algorithms used for pattern search are used to search for a particular pattern, whether a word, a sentence or several sentences in another, usually more comprehensive text. These algorithms have a wide range of applications. The most common use certainly includes online searches. Every reader of this thesis has used the Internet and searched for something on it. Even this operation could be considered as a use of pattern searching, where based on the user's input, words are searched for within the whole Internet, so a small number of search words are found in a huge set of words and based on the best matches the result is displayed. There may also be a large use of these algorithms in areas dealing with analysis. Algorithms for pattern searching can be categorized according to several parameters. [28]

2.2.1 Subdivision by Number of Patterns

Single Pattern Searching

During the comparison, only one pattern is worked with at a time. In the case of a match, it is saved and work with the next pattern starts. The selection does not necessarily have to be sequential. Some algorithms can prepare the patterns in advance and start working with the longest ones, for example.

- **Naive algorithm**, also known as brute force algorithm, is the simplest but most inefficient search algorithm. The way it works is that the pattern is "scrolled" through the text, looking to see if all the characters match, and if they do, it moves forward by one character (to avoid the end of the search) and looks for more matches in the same way. The time complexity is at worst θ(M(N−M+1)), where N is the length of the text, and M is the length of the search pattern. [29]
- **The Knuth-Morris-Pratt algorithm (KMP)** is more complex. In short, it can be said that the algorithm first preprocesses the string containing the patterns and creates an auxiliary field based on which it makes a decision. Thanks to this step, when a mismatch is found, it is possible to know which characters may follow the mismatch, and thus, there is no need to compare characters that are already known to be mismatched. The time complexity is at worst θ(N). [30]
- **The Boyer-Moore string search algorithm** also uses the same preprocessing of the patterning as the KMP algorithm. However, it uses the best case from the Naive algorithm and the best case from KMP, where based on these two heuristics, smaller arrays are created for each heuristic using the preprocessed array, where each pattern is shifted to the last possible position (Naive shifts the pattern by one position, KMP shifts it by several positions, Boyer-Moore shifts it by the maximum number of positions.) Then the best heuristic is chosen, and a match is found by pairing from the end. The time complexity is in the worst case θ(MN). [31]

Multiple Pattern Searching

During the looping, all the patterns (finite set) for which the step is searched are worked on at the same time.

- **Aho-Corasick algorithm** abbreviated AC also works on the principle of preprocessing of patterns. In the worst case, the time complexity is θ(N+M+Z), where Z is the number of words found. [32] The algorithm process can be divided into four steps:
  1. Creating trie – Trie is a data structure that stores the value information and information about its children. A trie is a similar structure to a binary tree, but the number of children of each node can be unlimited, unlike a binary tree where the children can only be 2. The second difference is that it works with chars, not numbers. Thanks to the trie, storing and searching data very efficiently is possible.
  2. Creating suffix (failure) links – Links used by matcher when a charac-

ter cannot follow a trie edge occurs. Their creation is best explained graphically. The images come from a presentation available at short-url.at/dlUZ2 [33]. For brevity, only the key states of suffix link creation will be highlighted.

In Fig. 2.1, there is a state slightly different from the rest of the algorithm: after creating a trie, all followers of the trie will automatically be assigned suffix lines referring to the root. That is followed by a loop through the second "row" of the trie. The trie is searched using a Breadth-first search (*BFS*). That is followed by entering the node with the character "a" more in Fig. 2.2.

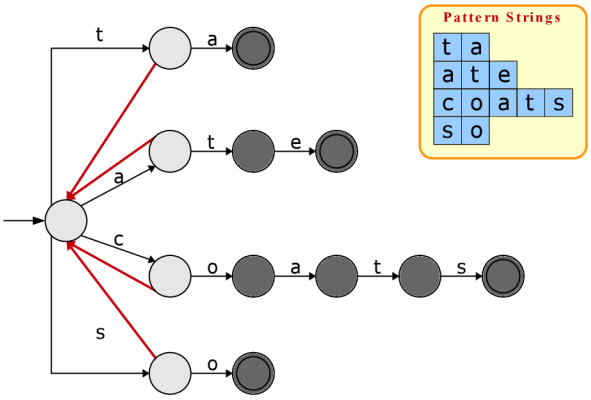

Fig. 2.1: Suffix links for root

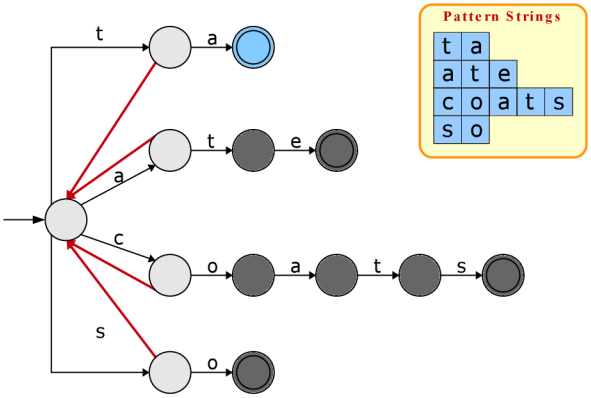

Fig. 2.2: Beginning of BFS, starting with the letter "a"

The Fig. 2.3, 2.4, 2.5, 2.6 show the algorithm flow. The algorithm asks if it is possible to get to the given node faster at a higher position (on the left in the picture). The node finds out that there is no way out of it, and using the suffix link, it reaches the root. Here it is asked if there is a node with the letter "a". It exists and lies in a higher layer than the original one, so a suffix link is created to it. If the node does not contain a follower with the letter "a", a suffix link would be created from the letter "a" from the second row directly to the root since there is no other row then the first.

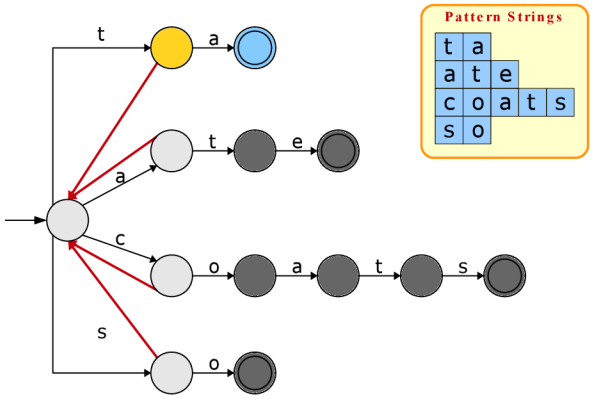

Fig. 2.3: Entry to parent node containing letter "t"

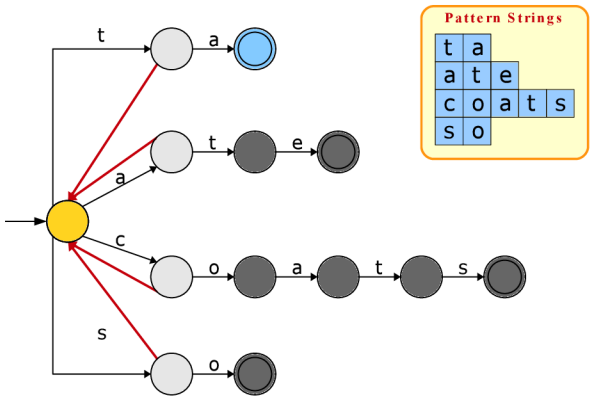

Fig. 2.4: entry to root node through suffix

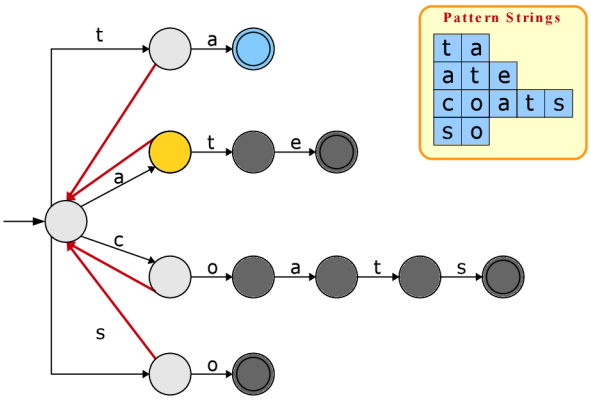

Fig. 2.5: Search if root contains node child with the letter "a", if yes, enter.

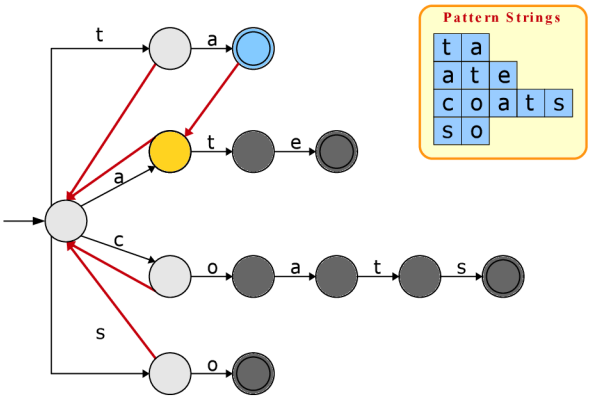

Fig. 2.6: Create suffix between found letters

If the letter "a" is in the third row, searching the second row would also be necessary. This will be partially outlined in the figures 2.7, 2.8, 2.9 and 2.10.

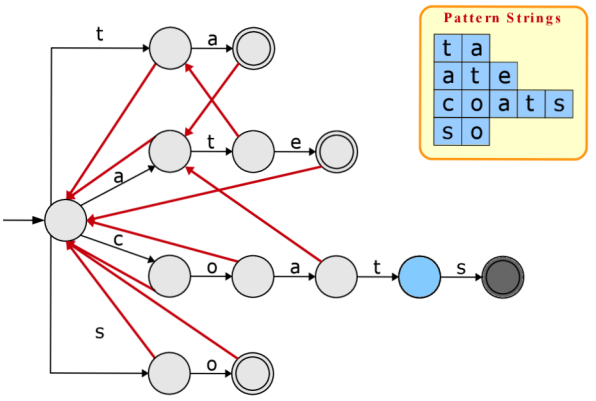

Fig. 2.7: Begininng with letter "t" from 4th row

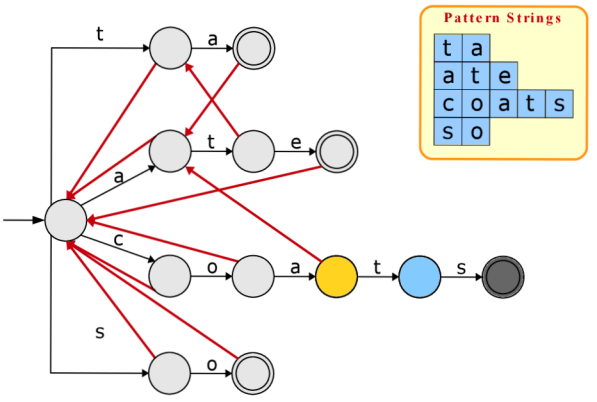

Fig. 2.8: entry to parent node containing the letter "a"

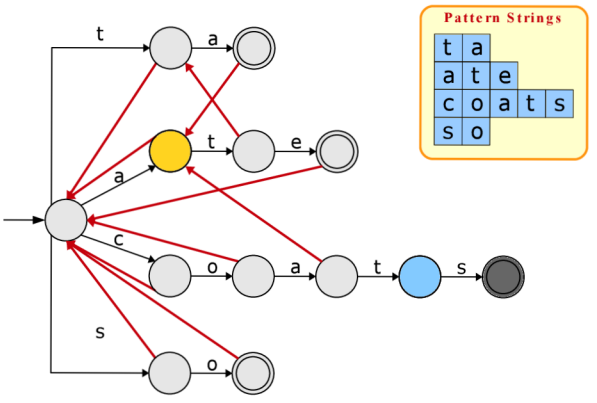

Fig. 2.9: entry to the node containing the letter "a" through suffix link

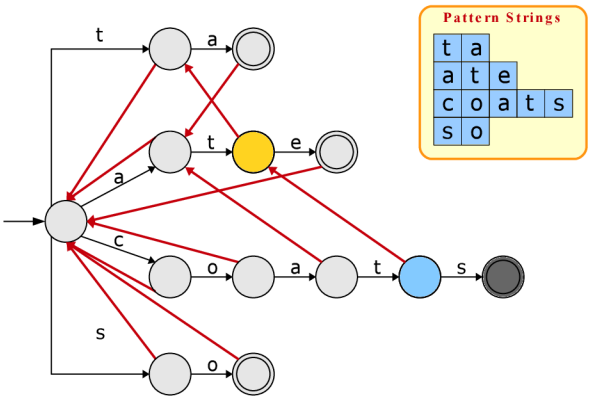

Fig. 2.10: Check if the black link contains the letter "t", if yes, create suffix link between letters

Fig. 2.7 refers to the state the automaton is in after performing the operations mentioned above and the sequential looping. The following figures show how the link between the letter "t" from row 4 and the letter "t" from row 2 was created.

According to the previous theory, the assumed state would be the entry into the root and, from there, the path to the letter "t" and the creation of the suffix line. However, there are more specified and efficient solutions. In this procedure, the algorithm would be swallowed, because of the long traversal, since the link of the closest possible node (series 2 is closer than series 1) is specified. Therefore, the first thing to do is always to use the black line input, and only in case of a mismatch is the suffix used and moved "higher".

3. Creating output links – Each node contains information whether it is final or not, this value is assigned to it on a simple principle, at the moment of adding a pattern to the automaton, the last character of each pattern is final, but the others are not. For example, in Fig. 2.7, the final nodes are marked as a double circle. Output lines are then created where a situation occurs where the suffix link points to the end node. That allows faster handling of patterns that are part of another pattern.
4. Implementing a matcher – Using sequential trie traversal, output all matches while using the output and suffix links. [32]

- **Commentz-Walter algorithm** consists of two very efficient algorithms, the Aho-Corasick mentioned above and the aforementioned Boyer-More algorithm. Commentz-Walter also works on the principle of trie and multiple pattern searches, supplemented by indexing. However, unlike Aho-Corasick, the trie

is reversed. The matches in this algorithm are made using the principle on which the Boyer-Moore algorithm works, which is end-to-end traversal, hence the reverse trie. [34] Commentz-Walter consists of two phases:

1. Creating a reverse trie – The procedure is the same as a trie, but the stamping works from "z" to "a" instead of from "a" to "z".
2. Matching phase – matching that works on the same principle as Aho-Corasick with a shifting technique derived from the Boyer-Moore algorithm. The algorithm scans the input pattern of the backwards and looks for when a mismatch occurs in the matching, and when it does, it stores this value in an index that is later used to compare with a pre-computed table created as part of the reverse trie creation. And thanks to it, it finds the distance needed to move the pattern to find the match. After this finding, the data and the index are saved. This data allows to find further matches better, thanks to the fact that the algorithm learns the positions of some characters. [34]

3 Implementation

This chapter will describe how the individual parts of this thesis were created, the problems and their solution and the final implementation. It will mention the creation of dictionaries, web scraping, the creation of the dictionary search algorithm in Java and the implementation of all these elements in the web application.

A diagram of how the automatic insertion of job ads will work is shown in Fig. 3.1.

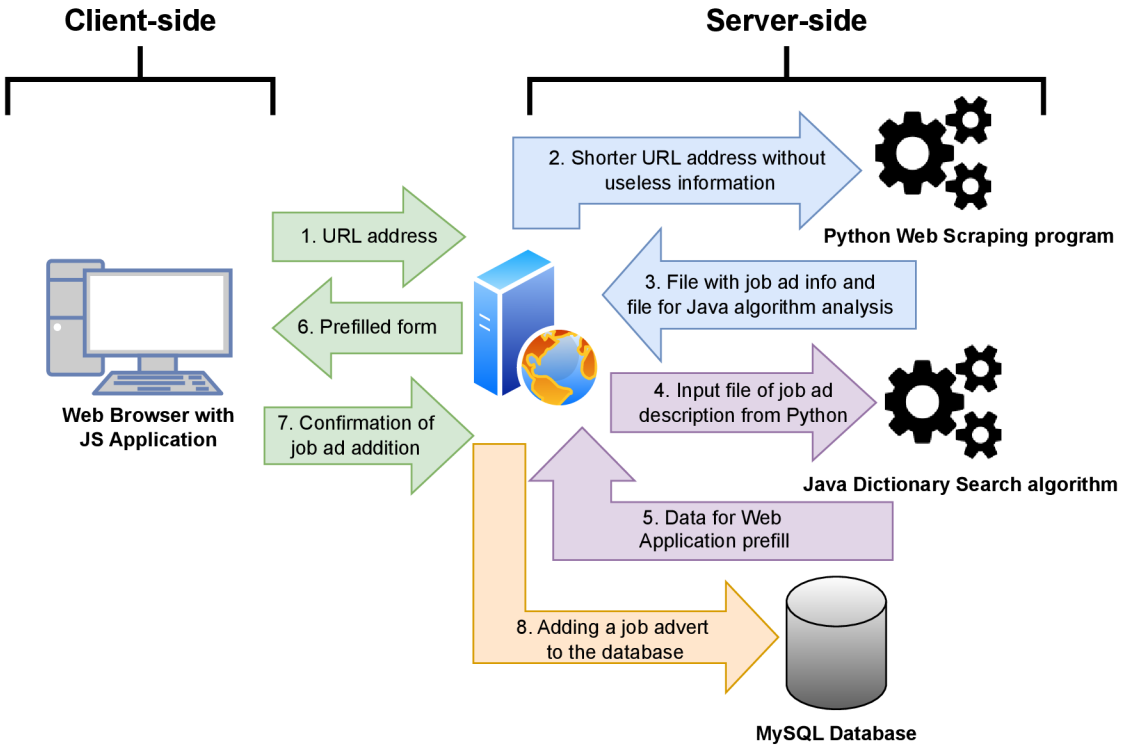

Fig. 3.1: Diagram of the automatic insertion of job adverts

3.1 Dictionary

The first step of this thesis was to create a keyword dictionary. The easiest way to create it was to use the Google Sheets web application. Google Sheets was chosen for the following reasons:

1. Easy access for multiple users who can edit the table/dictionary in real time after granting rights.
2. Access to data is possible anywhere and anytime, provided a functional internet connection exists.

3. Compatibility with all major operating systems.
4. The ability to download and work with the data in the format needed for the system.

What goes together with the advantage of working with data anytime and anywhere, which is possible thanks to the storage of data in the cloud storage, a drawback is the unavailability of data in the case of non-functional internet. In case of changes in the dictionary, either editing or adding new words and groups, it is necessary to download the dictionary and replace the old file with the new one. This problem does not affect this thesis in any way because the program works with data that have already been downloaded.

The dictionary is written horizontally to make the data easier to manipulate. The first entry of a given row is always the name of the Competency/Skill Group, and then within the row, the other keywords that fall into the category of the given Competency are written. To the already created structure provided by the thesis consultant and which contained the names of the Competencies and a couple of skills, it was necessary to add other keywords/skills related to the given Competency. In total, four ways were used to create the dictionary.

The first way was **Expert input**. REWIRE partners [35] proposed a preliminary dictionary with a few example words in every Competency and the table's basic structure.

The 2nd way was to select words from the definition provided by the thesis consultant at shorturl.at/jwAX0. This is based on the definition of **the ENISA (*European Union Agency for Cybersecurity)* framework**. Words were selected from ENISA key skills and knowledge descriptions. The terms were selected based on two parameters: the expression is unambiguous, and its use is assumed to correspond to the given Competency group; the word is likely to appear in the written text. For instance, the ENISA Cybersecurity Risk Manager profile contains "maturity models", "mitigate risks", "risk management tool", and "risk sharing options" that specifically identify the profile and can be added to the dictionary. The selected words were then bolded in the text for better orientation. The relationship between the ENISA competency framework and REWIRE Skills Groups can be seen in Fig. 3.2.

The third way was to select words, also from the material provided by the consultant of the thesis, specifically from the table at shorturl.at/zGKQW. This is based on the definition of **Competencies framework**, according to the **NICE** (National Initiative for Cybersecurity Education), which is part of the **NIST** (National Institute of Standards and Technology). As with the second way, the words were selected based on two parameters: unambiguity and expected occurrence in the text. For example, for group Data Security were found the words "confidentiality", "integrity",

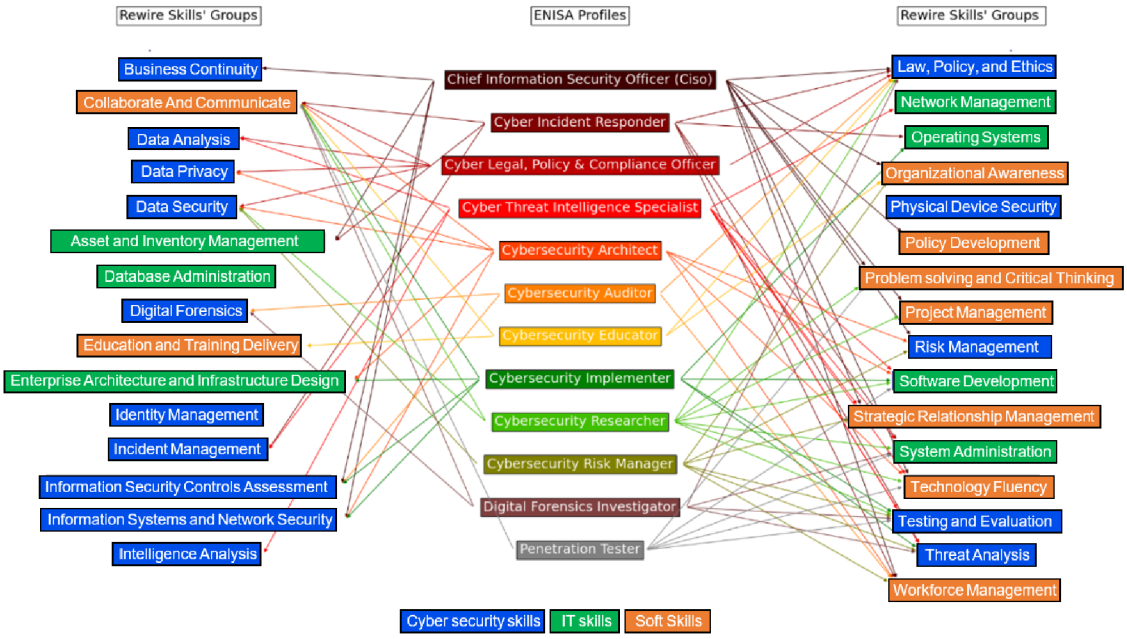

Fig. 3.2: All 31 REWIRE Skills Groups and their connection with ENISA and the categories to which they belong

and "availability" in the related Competency definitions. The selected words were then bolded in the text for better orientation.

The fourth and last way the keywords were selected and added was by inventing the words by the author as an **Expert knowledge**. Here, the way the words were created could be divided into subcategories, but the ways will be only briefly described for better readability. Some words were chosen according to the author's experience with the given Competency. For example, in the "Database administration" competency, the words PostgreSQL and phpmyadmin were added because the author knows them from previous years of studies. Different keywords were chosen based on the judgment that they could be useful with the given Competency. Other words were added on the basis of reading existing job ads. And last but not least, synonyms were used to already existing added words.

After the dictionary was created, the task was to distinguish which data already existed, which were added by the second, third method and which were created by the fourth method. In the created dictionary available at: shorturl.at/jvyP9, according to the following criteria:

1. Words already created in the table will be kept black.
2. Words created by method two will have the red colour.
3. Words created by method three will be blue.
4. Words created by the author, i.e. by method 4, will be marked green.

Intelligence Analysis	analytic tradecraft	technical analysis	technical reporting	feedback	customized analysis
Law, Policy, and Ethics	GDPR	General Data Protection Regulation	national regulations	international regulations	HIPAA

Fig. 3.3: Example of words in dictionary

While working on the other parts of this thesis (Web scraping and Java program), an updated version of ENISA definitions was delivered. It was necessary to modify this dictionary where a few old entries were deleted and new entries were added.

After the creation of the first dictionary, it was then necessary to test it with a program written in Java, the creation and functioning of which will be explained more in the next chapter. Testing was performed on 110 job advertisements that were already part of the database, and accuracy was established by comparing the output of the program with the data that were filled in by a human. The accuracy of this **Two-word Dictionary** was around 41%, which as an initial test was not the worst result, but still not quite optimal.

Therefore, a different method of dictionary creation was chosen, and this was a dictionary that consisted of only a one-word entry – **One-word Dictionary**. For example, the entry Risk management produced two entries, namely risk and management. However, from the initial dictionary were not selected such words that have meaning only in the case of their verbal name, such as data mining, Where, if there was a split of these words mean either something else non-relative to the group or such words do not exist at all. After completing this dictionary, another comparison was made on 110 job ads. Here the accuracy was much better, around 72%, but this accuracy was slightly misleading since its calculation only took into consideration the so-called True Positive cases, i.e. it was calculated from a simple relationship number of groups found/number of groups that should have been found and the calculation did not take into consideration the groups that were found by the program but should not have been found.

It was, therefore, necessary to use the calculation:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN},$$

where ACC stays for accuracy, TP for true positives, TN for true negatives, FP for false positives, and FN for false negatives.

Using this formula, even the result of the first dictionary would not have been so bad, i.e. the accuracy of 58.6% and for the newly created dictionary, it was a slightly better value of 59.6%

Based on the results, it was clear that working with the One-word Dictionary option would be better. The following dictionary had the task of refining this One-word Dictionary and getting rid of cases where groups were found that should not have been found. That was achieved by using the so-called "weights" of the dictionary. Within the new **Weighted Dictionary**, the One-word Dictionary was divided into two parts.

The first part contained words that, when found in the text, the group to which the word belonged was automatically set to 1 (meaning that this category was present in the text). The second part contained words that could mean the presence of a given group in the text but did not have to. If more words from this group were found, the occurrence of the assigned group would be more likely.

The essential element of this dictionary was the precision that the words from the second dictionary (Not 100% certainty of occurrence of the group) had to meet. In the first case, it was worked with the version that the number of words found must be at least 50% of the number of words in the given working competency. This idea was not wrong, but in categories where there were, for example, 14 or more words, rarely more than seven words were found. Therefore, a second approach to Weighted Dictionary was created, which was the method, if the total number of words in a given competency is less than 6, it works with 50% as in the previous case. If more words exist in a given Competency, only three are searched.

A comparison of these results can be seen in Tab. 3.1, which shows that the most accurate dictionary is the last mentioned Weighted Dictionary 2^b.

Name of Dictionary	TP (%)	ACC (%)
Two-word Dictionary	40.9	58.6
One-Word Dictionary	71.7	59.6
Weighted Dictionary 1^a	53.87	63.17
Weighted Dictionary 2^b	58.69	63.37

Tab. 3.1: Dictionaries results

^{a} match with half of the words in the group
^{b} match with three words in the group

3.2 Java Program

Another task of this thesis was to create a program that can find words in a long sequence of characters (sentences) that match the words from the dictionary, whose creation was described in the previous chapter.

Thus, studying and searching for which algorithm to choose for this task was necessary. Finally, thanks to the recommendation of the thesis consultant and the information found, two algorithms were selected, namely the Aho-Corasick and Commentz-Walter algorithms. These algorithms are some of the best search algorithms and therefore are believed to be ideal for this thesis. Reasons for choosing the Aho-Corasick algorithm:

1. Speed and efficiency – the main reason.
2. Ideal for dictionary-matching with long text.

The reasons for choosing the Commentz-Walter algorithm are similar:

1. Speed and efficiency – the main reason.
2. Ideal for dictionary-matching with a text.

Which algorithm to use in our case depends on several factors, namely:

1. The length of the dictionary.
2. The length of the text being scanned.
3. The number of hits found.
4. Their accuracy.

3.2.1 Aho-Corasick Algorithm

First, the focus will be on the creation/implementation of the Aho-Corasick algorithm. To begin with, it is worth saying that the algorithm is definitely not trivial and simple, and it may take more time to understand it than it seems. After its theoretical understanding, its creation could be divided into four steps.

1. Creating trie.
2. Creating suffix (failure) links.
3. Creating output links.
4. Implementing a matcher.

The first step was to create a trie data structure. The content of this structure, in this case, is the created dictionary. Originally the trie was made up of a few manually inserted words. Still, since the base seemed to work, the question arose as to how to efficiently get the individual words of the dictionary into the Java program and then use those words to generate the trie data structure. The Apache POI library available at https://poi.apache.org/ was used to get the data from Excel. This library allows working with columns, rows, and individual cells as needed, which is necessary for this thesis. It was used mostly to browse individual records by rows. Another handy feature is the ability to get the names of individual sheets. In the case of this thesis, it is the first possible sheet, which is not a problem. However, the problem could arise if the language of Excel was other than English. For example, in

the Czech language, the sheet's name is "list", and if the path were entered directly in the code, an error would occur, and the program would be broken.

After figuring out how to work with the library, it was possible to have the words successfully converted into a program by browsing through the individual cells. However, the words needed to be stored in some data structure in the program to make them easy to work with. In order to maintain some sort of link between the first word in the row, i.e. the name of the given Competency, and the other keywords, an ArrayList data structure was chosen, which contained another ArrayList containing the individual records. This more complex structure, however, is very efficient, and thanks to its properties, it is possible to keep the required groups together while accessing the records individually. Getting the data into the ArrayList of ArrayList data structure was not so simple. The moment the program tried to create a new record in the ArrayList for the cells that these contents were empty, errors occurred, and the program was not working. Initially, this problem was solved by the try-catch structure, but inserting empty cells was not at all convenient, so the problem had to be solved another way. Eventually, a simple solution was found, and that was to first save the loaded data into a helper variable, check if there are no empty cells, and only if the condition is met, the record was inserted into the ArrayList structure.

Listing 3.1: Loading data from Excel to ArrayList structure

```
ArrayList<ArrayList<String>> data = new ArrayList<~>();
//Can add any dictionary Excel file name.
String pathtable = String.valueOf(Paths.get(args[0]));
FileInputStream fileInputStream =
new FileInputStream(pathtable);
XSSFWorkbook xssfWorkbook =
new XSSFWorkbook(fileInputStream);
XSSFSheet sheet = xssfWorkbook.getSheetAt(0);
int rows = sheet.getLastRowNum();
int cells = sheet.getRow(1).getLastCellNum();
for (int r = 2; r <= rows; r++) {
    XSSFRow row = sheet.getRow(r);
    ArrayList<String> radek = new ArrayList<>();
    for (int c = 3; c <= cells - 1; c++) {
        XSSFCell cell = row.getCell(c);
        String value = cell.getStringCellValue();
        if (!value.equals("")) {
            radek.add(value);
```

```
            }
        }
        data.add(radek);
}
```

Thanks to the created structures, it was finally possible to create a trie structure by browsing individual records. It was already clear that the algorithm would not be simple, but the trie was finally created with the most significant probability based on the loaded data.

The second step was to create suffix (failure) links. Here the biggest problem of the thesis occurred due to the logical complexity and generality of the algorithm. Studying the algorithm, together with studying how to create the suffix links, took several days. Unfortunately, despite this, the author only achieved a theoretical understanding of the algorithm because the practical solution was really complicated and complex. Therefore, the author of the thesis could not create the algorithm. That is why a new approach was chosen in the thesis. After consultation with the supervisor and the consultant, the thesis was moved backwards. Due to the impossibility of creating the algorithm, the method chosen was to use an algorithm already created, and the aim was to transform the program into its own image.

After finding a few implementations of this algorithm, they were downloaded and tested to see if working with them according to the goals would be possible. Two implementations that looked really good were found.

The First Implementation

The first implementation, containing relatively short code, came from the geeksforgeeks.com website. Running this implementation was quite simple and intuitive. Changing and modifying it was, however, more complex. The implementation worked only with keywords which were directly written in a simple string in the program, and the keywords were not allowed to contain spaces, uppercase letters and various special characters. It was, therefore, necessary to modify the algorithm implementation to be able to use data from the already created ArrayList of ArrayLists structure or to work directly with the trie that had already been created.

Even before choosing one of the options, it was necessary to solve the problem with the limited ability of the algorithm to work only with small letters of the alphabet. This problem could be eliminated at the stage when the words from the dictionary were loaded into the program. For this purpose, the regex function was used, which allowed filtering according to the set parameters. After filtering, it was possible to save the data so that only minor alphabet characters were worked with.

As a first attempt, the option was chosen to try to work with the already created trie in the implementation. The code was not functional, even with the various changes that needed to be made in the code.

Therefore, the second option was chosen, which consisted in creating the trie with the algorithm that was already part of the implementation and changing only the data structure and the way of passing the keywords to other methods that are responsible for creating the trie, suffix links, output links and the subsequent matcher. However, this way of solving the problem did not work either.

So a third partial test option was offered, where the downloaded implementation was restored to its original state. Only the position containing the keywords was modified without code and structure modifications. To avoid inserting the words manually, a part of the code created earlier was used, which originally had a different purpose, but in the end, it worked perfectly for this purpose. After using the regex function and then saving it to the ArrayList of ArrayLists structure, it was only needed to display the resulting structure and format the output to match the same formatting as the input for inserting the keywords. The interference to the code was absolutely minimal, and from the point of logic view, it should be functional. However, despite these steps, the downloaded implementation did not work correctly.

The downloaded algorithm worked correctly only until the limit of 500 characters in the string containing the keywords was exceeded. This length was set as a constant with which other parts of the code worked, but even after rewriting it from 500 characters to the number 9500, which is necessary for our dictionary (of course, even with a margin-left), the algorithm did not work properly, and the implementation was discarded. So the other option was to start the second implementation.

The second implementation

With noticeably longer and more complex code, the second implementation comes from https://github.com/robert-bor/aho-corasick. The procedure of getting familiar with the finished implementation was similar to the previous implementation but also different in some points. The difference was, for example, that the second implementation did not contain an executable part of the code. It was necessary to create it, which was relatively easy, thanks to the knowledge acquired during the reinterpretation of the first implementation and the instructions included in the original work.

After studying the instructions, the `main` executable method was created, which contains the executable part of the code. The first methods used were the ones that are offered by the created code. These were used to determine what properties the search algorithm should have, such as the ability to use multi-word keywords, work

just with whole words, and not differentiate between upper and lower case, which is very good for this bachelor thesis. Usage of these can be seen in Listing 3.3.

After the features for working with data were selected, it was necessary to test the program. A few words from the dictionary were selected, these were manually inserted, and then a test text was inserted in which the words matching the words from the dictionary were to be found. The code did not contain any errors, so another part of the code was added to list the found words. That was also successful, and therefore it was possible to verify that the algorithm is indeed functional and finds all the keywords in the text.

Now it was necessary to make the most tricky part of this process work again, namely to make the algorithm be able to work with the whole dictionary, not just partial data.

Again, two options were offered to solve the problem. The first and more desirable option was to redesign the algorithm to be able to work with data from the ArrayList structure. That was not achieved in the end because of how the program works and reads data.

So the second option was used, which was to manually insert the words individually using the `.addkeyword`("word from dictionary") function. Due to the large number of words in the dictionary and the assumed possible expansion, manually inserting each word would be very time-consuming. For this reason, a small subprogram has been created to output the data loaded in the ArrayList of ArrayLists structure together with the necessary syntax so that the output provided by this subprogram can be used as input for the keyword addition part of the main program.

Listing 3.2: Sample of subprogram

```
for (int r = 2; r <= rows; r++) {
    XSSFRow row = sheet.getRow(r);
    for (int c = 4; c <= cells - 1; c++) {
        XSSFCell cell = row.getCell(c);
        String value = cell.getStringCellValue();
        if (!value.equals("")) {
            arr.add(value);
        }
    }
}
for (int i = 0; i < arr.size(); i++) {
System.out.println(".addKeyword(\""+arr.get(i)+"\")");
}
```

Listing 3.3: Output of subprogram

```
.addKeyword("exercises")
.addKeyword("courses")
.addKeyword("learn")
.addKeyword("learns")
.addKeyword("education")
.addKeyword("educate")
.addKeyword("workshops")
.addKeyword("mentor")
.addKeyword("guide")
.addKeyword("course")
```

This part has to be done manually, so whenever there is a change in the dictionary, it is necessary to download it, replace the old dictionary, run the subprogram and insert the generated output into the main program. The program works this way because it is assumed that the changes in the dictionary will not be persistent. After the thesis is finished, it is possible that the dictionary will remain the same, and therefore not so much emphasis will be put on these steps. The keywords were successfully loaded, and it was possible to move on with the program.

Listing 3.4: Creating trie and usage of subprogram

```
Trie trie;
{
trie = Trie.builder().onlyWholeWords().ignoreCase()
         .ignoreOverlaps()
     .addKeyword("BCP")
     .addKeyword("BCM")
     .addKeyword("business")
     .addKeyword("prevent")
     .addKeyword("improvisation")
     .addKeyword("prevention")
     .addKeyword("continuity")
     .addKeyword("improvise")
     .addKeyword("processing")
     .addKeyword("cluster")
     .addKeyword("anomaly")
}
```

The program was finally functional, and therefore it was possible to start making the modifications that would be needed to work with the program within the web application. The program has so far only worked with a string (a sequence of

sentences) that was manually inserted. In order to implement the program and use it for work in a web application, it is necessary to be able to read this data without physical interaction with the program. For this reason, the program will use the reading from the file. The principle of operation will be straightforward, using a web scraper, the data will be saved into a file, which will then be passed to the program to find the corresponding words. It will read it, process it and then run the matching algorithm.

The last necessary step of this program was to format the resulting data. That will allow better and easier manipulation of the data provided by the program in the subsequent work. The data that will be needed in the output are:

1. Words found matching the dictionary.
2. Position at which the found word starts and ends.
3. Names of Competencies and the number of matching words they contain.

The 1st and 2nd points are made possible by a data structure that was already part of the algorithm when its implementation was downloaded, so it was only necessary to list it.

Point 3 was achieved in the following way. To be able to work with words individually and without the part containing their position, the regex function was used to get rid of the part containing the position of the word. Then the split function was executed, which split the resulting text string into individual words, where the splitting element was the comma character.

Finally, the individual filtered words were stored in the array structure. The initial focus was to create several counters, and then in the process of browsing and comparing the ArrayList, made up of words from the dictionary and an array made up of words that match. When there is a match, the record representing the first position of the given row (name of the given Competency) is always queried for what position it is and on the basis of the switch structure, the given counters would be raised by one. That would not be an optimal solution because it would be necessary to create more than 30 counters, where each one would have a different name, and if it were not used, it would take up memory space. Creating more than 30 conditions in the switch structure would also be necessary, and the result would be printed as 30+ records with minimal changes.

This solution was replaced, in a much simpler way, by creating another array with the size of the number of competencies and using the same index value in the case of a match between the ArrayList and the array to increase the counter for a given category – later this way of counting words a little bit changed.

Due to the creation of new Weighted Dictionaries, the approach of counting the number of words found has been changed to the logic: at least one word from the category of 100 percent words occurs here, and the given group is set to 1.

In case a word from another dictionary (not 100%) occurs in the text, where it works with percentage accuracy or a number of words, the value 0 and 1 is chosen based on the given criterion.

Listing 3.5: Output of created program

```
[0:12=improvisation, 25:31=cluster, ...]
Number of words from Business Continuity is: 4
Number of words from Data Analysis is: 1
Number of words from Data Privacy is: 5
Number of words from Data Security is: 2
Number of words from Digital Forensics is: 1
Number of words from Identity Management is: 2
Number of words from Incident Management is: 2
```

Listing 3.6: Output of created program after rework

```
Number of words from Business Continuity is: 1
Number of words from Data Analysis is: 1
Number of words from Data Privacy is: 0
Number of words from Data Security is: 0
Number of words from Digital Forensics is: 1
Number of words from Identity Management is: 1
Number of words from Incident Management is: 0
```

After completing these steps, it was only necessary to save these results to a file so that they could be worked with in the web application. The most suitable format for working within the web application was the json format. The step to save the data in json format was easily achieved by the Gson library, which greatly facilitated the work of formatting the output.

A `.jar` file was then exported from the program, which can be run from the web application using the command line.

3.2.2 Commetz-Walter Algorithm

For the Commentz-Walter algorithm, also due to the complexity of the algorithm, the method of not creating the algorithm was chosen, and only the existing implementation will be downloaded and then modified. Unfortunately, there was an even bigger problem here than with the Aho-Corasick algorithm. The author could not find any working implementation of this algorithm anywhere. Only a program written in Python and C was found. It is available at shorturl.at/nqZ29. Despite this, an effort was made to get this program working. To see if the algorithm works.

After a little study, the program was finally able to run. Unfortunately, the purpose of the program is not to list the words found but to compare the Aho-Corasick and Commentz-Walter algorithms. Algorithm Commentz-Walter, whose output would be matching words like the Aho-Corasick algorithm, could not be created because of its difficulty.

Therefore, at least the downloaded implementation was used to see which of the two algorithms is faster. One random text from a real job advertisement was chosen as the searched text. The words from the dictionary were all inserted successfully using a file containing all the keywords separated by commas. However, the results are rather indicative, as the algorithm used in the comparative implementation may be different from the algorithm used in the bachelor thesis. Which algorithm is faster will be discussed in the summary of the thesis.

3.3 Python Script

After creating and testing the dictionaries, the Java dictionary search algorithm Java, it was necessary to create the a Python script for web scraping.

Since the previous two tasks have already been completed and tested, it was necessary to create a script that will allow, after entering the URL link, to download the content of this web page or important parts of this content and save this data to a file that will serve as an input file for the Dictionary search algorithm.

In the beginning, there was an effort to do this task (Web scraping) without using other scripts or tools and ideally to create it directly in PHP, which is a part of the web application, but after several unsuccessful attempts to try methods and tools such as cURL, file_get_contents, Guzzle, Selenium, the option of using Python script was chosen, because it was much easier to create and develop.

The first method to be considered was one that would use the "cURL" library for its work because of its speed and low error rate. However, there was a misunderstanding of the functioning of the "cURL" library, which allows downloading the source code of the page, with which further text modifications were planned (stripping the text of unnecessary data) so that it was possible to make this method globally for all web pages. Still, unfortunately, this method can only work with static pages, i.e. the cURL library downloads the page's HTML code and then works with it. However, if the embedded web page is written dynamically (using JavaScript, PHP, etc.), the cURL method does not see this content.

After a short research, the Selenium library was tried again, not in PHP code, but as a Python script. After understanding how the library works in Python and the initial parameter settings, this way of scraping was tried and worked. Then it was necessary to specify how to select the web elements in the script. Selenium

allows searching for parts of an element according to different criteria, such as the element's class name, the element's id, the element, the name of the element, and the XPath expression. The most promising for this task was the method of searching by element id, the advantage of which would be that even if the web page of the advertisement were changed, its id would probably remain the same. Unfortunately, this method did not work in the final implementation or took too long. Therefore, the XPath expression method was chosen, simplifying the whole path to the element according to the site's structure. The xpath path can look like this: `/html/body/main/section[1]/div/div/section[1]/div/div/section/div`. It has one disadvantage: the moment the author of the web page from which the web advertisements will be drawn changes the structure of this site, it will be necessary to perform the mapping again. The advantage of this method is that the search for elements is faster and almost error-free.

Within the script, search methods by tag or class name were also used to automatically fill in data within the web application, such as job ad name, company name, location and others.

Listing 3.7: Sample of Selenium maping

```
driver.get(url)
job_description = driver.find_element(By.XPATH,
"/html/body/main/section[1]/div/div/section[1]/div/div/
section/div").get_attribute("innerText")
job_title = driver.find_element(By.TAG_NAME,"h1").text
job_company = driver.find_element(By.CLASS_NAME,
"topcard__flavor").text
countryCityProvince = driver.find_element(By.XPATH,
"/html/body/main/section[1]/div/section[2]/div/div[1]/
div/h4/div[1]/span[2]"
).get_attribute("innerText").strip()
driver.quit()
```

After creating these locators, it was also necessary to change the Selenium settings so that the browser opened by this script has the parameters language=English and is headless, which means that the browser is not opened on the screen. Still, the action is performed in the background.

Then it was only needed to save the obtained files to a file and set the input parameter to a variable that always contains the URL address the user specifies.

Due to the different security of web pages and applications against server overloading and overloading, it was not possible to create a method that would be able to get from any URL the information necessary for the Dictionary search algorithm.

Therefore, creating a "mapping" within a Python script for each page from which advertisements will be added to the web application is necessary. Currently, the main source of job ads is Linkedin, which is also mapped.

3.4 Web Implementation

After completing all the tasks, using them in the web application was sufficient. After familiarizing with the web application and performing the necessary changes to get the application running on the localhost, modifications were made to the PHP code, where a new file named `analyzeJob.php` was created, which is invoked after inserting the URL within the web application and pressing the Analyze button. The main part of this PHP code is in a function so that the `return` function can be used here in case an error happens, allowing to return from the function back to the main code. It simply allows not to continue the code when an error occurs, and at the same time, this solution provides a simpler and clearer structure than if there was a branch of nested `if else`.

The first step is to check with php whether the embedded link belongs to a group of mapped sites (in this thesis's case, only LinkedIn) or not.

Listing 3.8: URL Check

```
$url = $obj->url;
$pattern = "/LinkedIn/";
if (!preg_match($pattern, $url)) {
    $output->error = true;
    $output->message = "Sorry, you need add your job
    manually, please insert linkedin job to analyze ";
    return;
    }
```

Listing 3.9: Running a Python script and example of error handling

```
$command = "cd ".$scrape_path." && python main.py ".$url;
exec($command, $outputText, $resultCode);
if ($resultCode != 0){
    $output->error = true;
    $output->message = "Error: Exec Python ended with
    error code".$resultCode;
    return;
}
```

Next, the redundant information in the embedded URL link will be removed more in Listing 3.8. That is followed by running the Python script with the cleaned URL as an input parameter more in Listing 3.9.

The output of the Python script is two files, one that serves as input for the Java program, which contains the necessary information for analysis (Job Description), and the other contains information about the job advertisement, which will be needed in the automatic filling in the web application. In case everything goes well, the Dictionary search algorithm will be executed by running the `.jar` file. If there is no error again, the result is a `.json` file containing the words found and a list of workgroups with values 1 and 0 (whether the workgroups appear in the text or not). These values are then retrieved within the PHP code using the `file_get_contents` method and converted into an array using `json_decode`. They are then processed using simple indexing such as `numbers[1]`. All obtained information (information about the job advertisement and individual values of given groups) is then loaded into a formula displayed in the web application after running the `analyzeJob.php` file. Part of the final formula can be seen in Listing 3.10.

Listing 3.10: Entering information into the form

```
$job_title = trim(str_replace(array("\r", "\n"), "",
             $infos[0]));
$resultForm = array(
    "title" => $job_title,
    "country" => $job_country,
    "cs_business_continuity" => $numbers[0],
    "cs_data_analysis" => $numbers[1],
    "cs_data_privacy" => $numbers[2],
    )
```

After creating the part of the backend mentioned above, it was enough to write/prefill the data within the frontend written in ReactJS (JavaScript) into the web form that is displayed to the user and was originally used for manual input. The result of the successful analysis can be seen in the Fig. 3.4 and 3.5, where the user just inserted the URL link and pressed the "Analyze" button and all the data was automatically filled in for him.

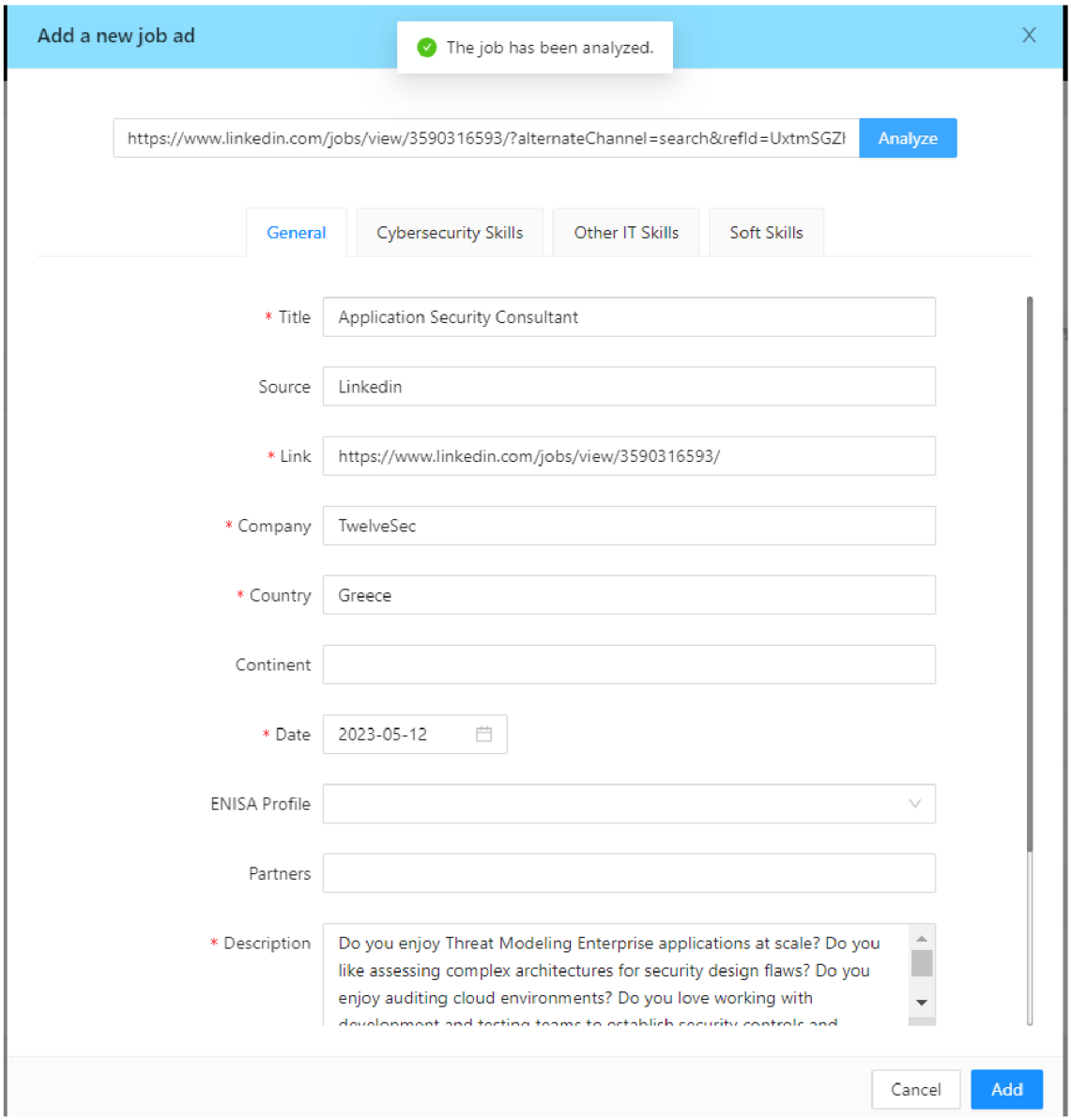

Fig. 3.4: Example of filling in advert details

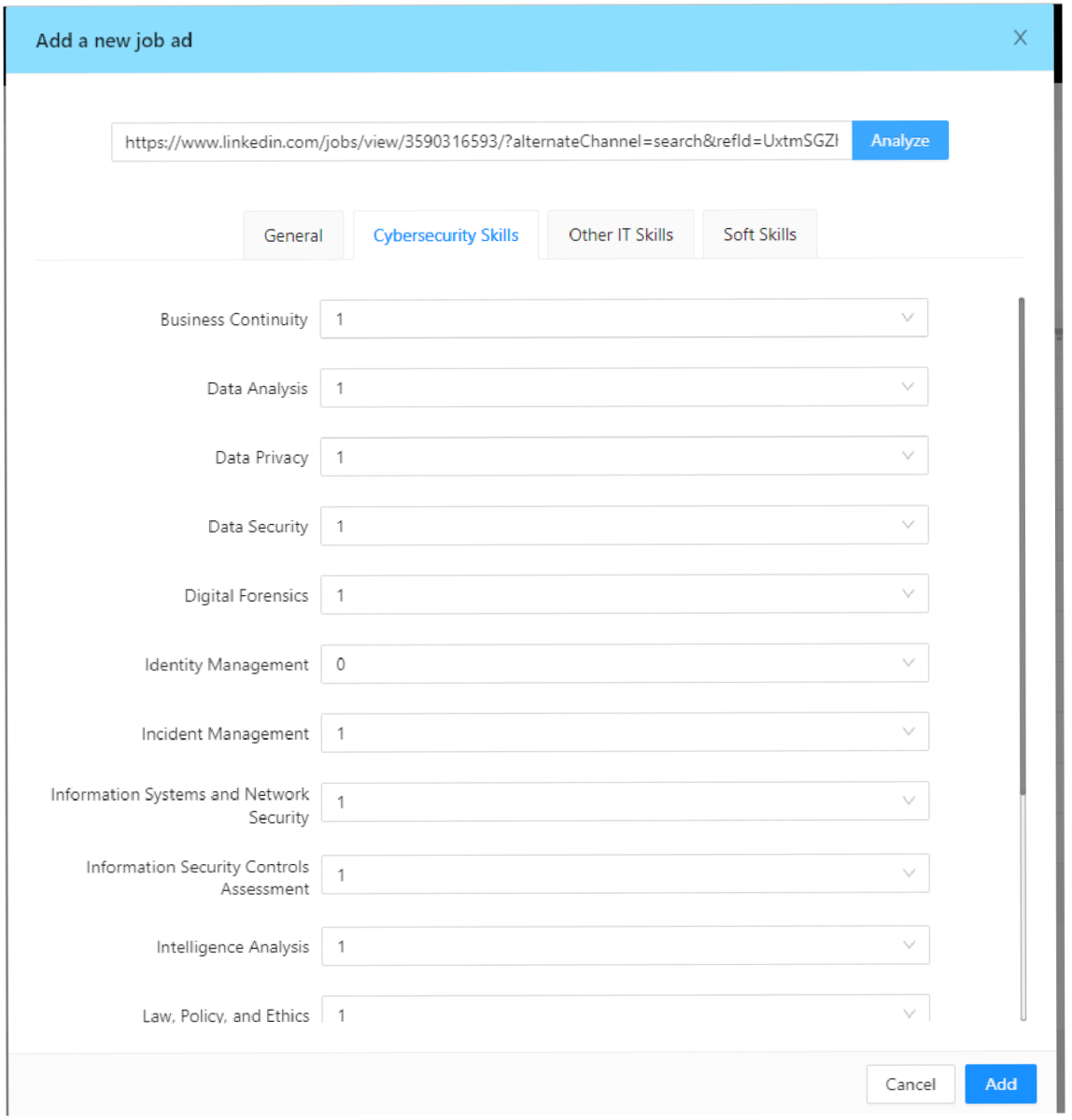

Fig. 3.5: Example of filling in details of occurring Skills Groups

Summary

The task of this bachelor thesis was creating automation for entering new job advertisements with dictionaries, a program to search for keywords in a large text and a web scraping script. The dictionary was created based on the definitions and ideas successfully. Unfortunately, the program using at least two algorithms and then comparing them has not been created.

Fortunately, at least one algorithm has been created and made operational, which allowed continuing with the thesis in the following steps, thanks to the data provided by the algorithm. Since it was impossible to create the second algorithm or find its ready implementation on the internet, at least a theoretical comparison was made.

Algorithms can be compared on the basis of several criteria. One of the comparisons can be a comparison based on the worst-case algorithm, where the Aho-Corasick algorithm has a worst-case complexity of θ(M+N+Z), where M is the length of the dictionary, N is the length of the scrolled text, and Z is the number of matches found. The Commentz-Walter algorithm has the worst case θ(MN). In the case that the worst case occurs, the Aho-Corasick algorithm is clearly faster. The average values occurring in this thesis are M=8500, N=3000. In this thesis, the worst case for Commentz-Walter should not occur. It occurs when all the characters of the pattern and the text are the same. For example, a pattern of five characters "a" is searched in a text containing 15 consecutive characters "a". Therefore, this indicator is not completely informative.

Based on the study [36], Commentz-Walter is faster in cases where the length of the patterns is longer. The span of a long pattern cannot be determined directly, but patterns that contain more than 20 characters can be considered a kind of breakpoint. The range of 20-120 characters per pattern is given as the span of a long pattern. There is no guarantee that once the length of 20 characters is reached, Commentz-Walter is automatically better. It is more of a break when its superiority may start to be felt, but depending on the character of the work, this may only become apparent, for example, when the pattern contains more than 40 characters.

Another critical point is the number of patterns, as the Commentz-Walter algorithm becomes less efficient when working with more than 13 patterns. That does not mean that Commentz-Walter works efficiently with only 13 patterns. However, rather than decreasing efficiency as the number of patterns increases, it is still a perfect search algorithm.

This thesis has patterns with an average length of 8.2 characters (One-Word Dictionary), and the current number of patterns is 515. According to theory, the Aho-Corasick and Commentz-Walter algorithms should be approximately equally

fast. Perhaps the Aho-Corasick algorithm should be slightly faster. That can be partially verified, thanks to the implementation mentioned in the section Implementation of the Commentz-Walter algorithm. See Listing 3.11.

Listing 3.11: Results of Aho-Corasick and Commentz-Walter algorithm comparison for one of the real job.

```
####     AHO-CORASICK     ####
ELAPSED TIME: 0.33917236328125
TOTAL MATCHES: 20
####     COMMENTZ-WALTER     ####
ELAPSED TIME: 0.39746546745300293
TOTAL MATCHES: 20
```

Therefore, the hypothesis that algorithms run for similar times and that Aho-Corasick may be slightly faster is correct. It still needs to be considered that the comparison is only theoretical, and the implementations of the Aho-Corasick algorithms may differ in each implementation.

In the dictionary selection, there were two options, the first was the Weighted Dictionary 2^b, and the second was the One-Word Dictionary. In case the user interaction was not considered anymore, probably the better option would be a Weighted Dictionary 2^b because of its highest accuracy of all the dictionaries tested, i.e. accuracy 63.37% calculated from the relation

$$ACC = \frac{TP + TN}{TP + TN + FP + FN},$$

where ACC stays for accuracy, TP for true positives, TN for true negatives, FP for false positives, and FN for false negatives.

In the context of this thesis, however, the possibility of highlighting the found words when entering a new job advertisement in the given categories is envisaged, where the better option is to find as many words as possible from the dictionary in the text and leave it to the user to decide whether the found terms really match the given job competency. This option is provided by the second One-Word Dictionary, whose True Positive accuracy value is the highest of all at 71.7%.

After choosing to use Python scripts, the part dealing with web scraping was also successfully completed. And the implementation itself, after tuning various details, worked on several devices, which meant its functionality. In the scope of the bachelor's thesis, it was not possible to test whether the finished automation is also functional in the web application, as it is already running in live operation. The implementation is currently planned for the summer.

Conclusion

In the scope of this bachelor thesis, it was successfully managed to create an automation for inserting new job advertisements containing the analysis of job competencies in the framework of an existing web application.

Due to the requirement for fast searching, the Dictionary multipattern searching algorithm Aho-Corasick was chosen as the most suitable for this thesis (due to the large number of words in the dictionary and their shorter length).

One-Word Dictionary was chosen as input data for this program, although the Weighted Dictionary 2^b had better accuracy. This dictionary was chosen because its True Positive accuracy was better than the other dictionary. This method is chosen because it is desired to find as many Skill Groups as possible, and then the user can easily choose whether the found Skill Group matches or not.

For the possibility of getting data from the embedded URL of the link, a Python script using the Selenium library was used, which allowed to get input data for the Dictionary search algorithm and also information about individual advertisements, which will be automatically filled in after the link is inserted in the web application.

All these steps are then used/run by a PHP file that runs and is executed on the server side.

As part of further work on the thesis, it is planned to add highlighting of found words in the web application, adding mapping for other pages containing web advertisements. And also implementation of the created thesis into a running website during the summer holidays.

Bibliography

[1] DOUPAL, F.: *Nedostatek zaměstnanců v ICT trvá. České uchazeče loví i zahraniční firmy* [online]. Last updated 14. 9. 2022. URL: <https://shorturl.at/iVX15> [Visited 5. 5. 2023].

[2] KOĎOUSKOVÁ, B.: *Web, webová stránka a webová aplikace, v čem je rozdíl?* [online]. Last updated 12. 8. 2020. URL: <https://www.rascasone.com/cs/blog/web-webova-aplikace-rozdil> [Visited 16. 11. 2022].

[3] *Web application* [online]. Last updated 31. 9. 2019. URL: <https://www.computerhope.com/jargon/w/web-application.htm> [Visited 16. 11. 2022].

[4] *Word Wide Web* [online]. Last updated 3. 11. 2022. URL: <https://en.wikipedia.org/wiki/World_Wide_Web> [Visited 16. 11. 2022].

[5] KOĎOUSKOVÁ, B.: *HTML PRO ZAČÁTEČNÍKY ANEB JAK ZAČÍT PSÁT WEB* [online]. Last updated 15. 7. 2021. URL: <https://www.rascasone.com/cs/blog/html-pro-zacatecniky-jak-psat-web> [Visited 16. 11. 2022].

[6] *HTML Standard* [online]. Last updated 8. 11. 2022. URL: <https://html.spec.whatwg.org/multipage/introduction.html#introduction> [Visited 16. 11. 2022].

[7] *HTML5 Differences from HTML4* [online]. Last updated 9. 12. 2014. URL: <https://www.w3.org/TR/html5-diff/> [Visited 16. 11. 2022].

[8] *HTML 5.2 IS DONE, HTML 5.3 IS COMING* [online]. Last updated 14. 12. 2017. URL: <https://www.w3.org/blog/2017/12/html-5-2-is-done-html-5-3-is-coming/> [Visited 16. 11. 2022].

[9] PEDAMKAR, P.: *Versions of Html* [online]. Last updated 2022. URL: <https://www.educba.com/versions-of-html/> [Visited 16. 11. 2022].

[10] *CSS: Cascading Style Sheets* [online]. Last updated 25. 9. 2022. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> [Visited 16. 11. 2022].

[11] BOS, B.: *A brief history of CSS until 2016* [online]. Last updated 17. 12. 2016. URL: <https://www.w3.org/Style/CSS20/history.html> [Visited 16. 11. 2022].

[12] BOS, B.: *What is PHP?* [online]. Last updated 2022. URL: <https://codeinstitute.net/global/blog/what-is-php-programming/> [Visited 16. 11. 2022].

[13] JACKSON, P.: *What is PHP? Write your first PHP Program* [online]. Last updated 3. 10. 2022. URL: <https://www.guru99.com/what-is-php-first-php-program.html> [Visited 16. 11. 2022].

[14] *WHAT IS SQL?* [online]. 2022. URL: <https://www.sqlcourse.com/beginner-course/what-is-sql/> [Visited 16. 11. 2022].

[15] KOĎOUSKOVÁ, B.: *JAVASCRIPT PRO ZAČÁTEČNÍKY: CO TO JE A JAK FUNGUJE* [online]. Last updated 28. 1. 2022. URL: <https://www.rascasone.com/cs/blog/co-je-javascript-pro-zacatecniky?fbclid=IwAR3zM5QvSfy19T2YXZRwcNiDPWgvgFu291BRdS5_S76evIE4VYZR3JRwJpU> [Visited 18. 11. 2022].

[16] JORDANA, A.: *What Is JavaScript? A Basic Introduction to JS for Beginners* [online]. Last updated 27. 10. 2022. URL: <https://www.hostinger.com/tutorials/what-is-javascript#3_Interactive_Behavior_on_Websites> [Visited 18. 11. 2022].

[17] HERBERT, D.: *What is React.js? (Uses, Examples, & More)* [online]. Last updated 27. 6. 2022. URL: <https://blog.hubspot.com/website/react-js> [Visited 19. 11. 2022].

[18] *Introduction to the DOM* [online]. Last updated 29. 9. 2022. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction> [Visited 19. 11. 2022].

[19] DESHPANDE, CH.: *The Best Guide to Know What Is React* [online]. Last updated 23. 10. 2022. URL: <https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs> [Visited 19. 11. 2022].

[20] *What is Java?* [online]. URL: <https://aws.amazon.com/what-is/java/> [Visited 22. 04. 2023].

[21] FASRIN, A.: *Let's Understand Java* [online]. Last updated 28. 10. 2021. URL: <https://medium.com/nerd-for-tech/lets-understand-java-261b2e6bcf2e> [Visited 22. 04. 2023].

[22] *Pros and Cons of Java | Advantages and Disadvantages of Java* [online]. URL: <https://data-flair.training/blogs/pros-and-cons-of-java/> [Visited 22. 04. 2023].

[23] *What is Python? Executive Summary* [online]. URL: <https://www.python.org/doc/essays/blurb/> [Visited 22. 04. 2023].

[24] KARANI, D.: *How does Python work?* [online]. Last updated 9. 1. 2020. URL: <https://towardsdatascience.com/how-does-python-work-6f21fd197888> [Visited 22. 04. 2023].

[25] BHAMRA, P.: *What is Selenium? Introduction to Selenium Automation Testing* [online]. Last updated 19. 4. 2023. URL: <https://intellipaat.com/blog/tutorial/selenium-tutorial/introduction/> [Visited 22. 04. 2023].

[26] TWIN, A.: *What Is Data Mining? How It Works, Benefits, Techniques, and Examples* [online]. Last updated 2. 8. 2022. URL: <https://www.investopedia.com/terms/d/datamining.asp> [Visited 19. 11. 2022].

[27] RŮŽIČKOVÁ, M.: *Data mining – Co? Jak? K čemu?* [online]. Last updated 14. 5. 2018. URL: <https://medium.com/edtech-kisk/data-mining-co-jak-k-ÄDemu-c5176179303b> [Visited 19. 11. 2022].

[28] *Applications of String Matching Algorithms* [online]. Last updated 7. 11. 2022. URL: <https://www.geeksforgeeks.org/applications-of-string-matching-algorithms/?ref=gcse&fbclid=IwAR3JztbCFHX6CPLE1vfVp9GC_cC5C7B_HuJ_jcnLmzPd0Nz36-jnFLna2H0> [Visited 20. 11. 2022].

[29] *Naive algorithm for Pattern Searching* [online]. Last updated 22. 8. 2022. URL: <https://www.geeksforgeeks.org/naive-algorithm-for-pattern-searching/> [Visited 20. 11. 2022].

[30] *KMP Algorithm for Pattern Searching* [online]. Last updated 18. 11. 2022. URL: <https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/> [Visited 20. 11. 2022].

[31] *Boyer Moore Algorithm for Pattern Searching* [online]. Last updated 9. 11. 2022. URL: <https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/> [Visited 20. 11. 2022].

[32] *Aho-Corasick Algorithm for Pattern Searching* [online]. Last updated 14. 6. 2022. URL: <https://www.geeksforgeeks.org/aho-corasick-algorithm-pattern-searching/> [Visited 20. 11. 2022].

[33] *Aho-Corasick Automata* [online]. URL: <http://web.stanford.edu/class/archive/cs/cs166/cs166.1166/lectures/02/Slides02.pdf> [Visited 21. 11. 2022].

[34] *Commentz-Walter algorithm* [online]. Last updated 19. 10. 2022. URL: <https://en.wikipedia.org/wiki/Commentz-Walter_algorithm> [Visited 20. 11. 2022].

[35] REWIRE - Cybersecurity Skills Alliance. *REWIRE Project Results* [online]. URL: <https://REWIREproject.eu/results/> [Visited 21. 11. 2022].

[36] DEWASURENDRA, S., VIDANAGAMACHCHI, S.: *Average time complexity analysis of Commentz-Walter algorithm.* Journal of the National Science Foundation of Sri Lanka, Sri Lanka: 2018. URL: <http://doi.org/10.4038/jnsfsr.v46i4.8630> [Visited 23. 11. 2022].

Symbols and abbreviations

URL Uniform Resource Locator
WWW World Wide Web
HTTP Hypertext Transfer Protocol
CERN European Organization for Nuclear Research
W3C World Wide Web Consortium
XML Extensible Markup Language
HTML Hypertext Markup Language
XHTML Extensible Hypertext Markup Language
CSS Cascading Style Sheet
PHP Hypertext Preprocessor
SQL Structured Query Language
SEO Search Engine Optimization
DOM Document Object Model
MVC Model View Controller
KMP Knuth-Morris-Pratt
AC Aho-Corasick
BFS Breadth-First Search
NICE National Initiative for Cybersecurity Education
NIST National Institute of Standards and Technology
ENISA European Union Agency for Cybersecurity
IoT Internet of Things
JVM Java Virtual Machine

List of appendices

A Appendencies

A.1 Installation Manual

The **first step** to get started is downloading, running and configuring the XAMPP bundle, that contains an Apache web server with PHP, a MySQL database server, an FTP server and other components. This software allows you to easily install and run all the necessary components in one step. Follow these steps:

1. On the `https://www.apachefriends.org/download.html` website, download the version for the current OS. In the case of this thesis, Linux version 8.0.28.
2. After downloading, open the terminal, change the rights for this file and run the program.
   ```
   $ cd Downloads
   $ chmod 755 xampp-linux-*-installer.run
   $ sudo ./xampp-linux-*-installer.run
   ```
3. The installation file will be started, continue with the Forward button, install the "XAMPP Core Files" and the "XAMPP Developer Files", and continue the installation by clicking Forward at the end Finish.
4. After the successful installation of XAMPP, run the following command:
   `$ sudo /opt/lampp/./manager-linux-x64.run`
5. This will open the graphical interface of XAMPP.
6. Under Manage Servers, click Apache Web Server and click the button on the right Configure -> Open Conf File -> Yes.
7. At lines 173 and 174, change user name and group from daemon to actual user, for example, john (actual user is shown in the terminal command line in front of "@" symbol, for example, `john@ubuntu`.
8. Save, close and start the localhost server using Start All button.

The **second step** will be to update and install all necessary packages. Open the terminal and add those commands:

```
$ sudo add-apt-repository universe
$ sudo apt update
$ sudo apt install python3
$ sudo apt install python3-pip
$ sudo pip install selenium
$ sudo apt install default-jdk
$ sudo apt-get install -y unzip xvfb libxi6 libgconf-2-4
$ sudo pip3 install requests
$ sudo apt-get install chromium-chromedriver
```

After completing the **previous 2 steps**, it is possible to add documents created in this thesis, specifically to the `/opt/lampp/htdocs` folder. After inserting the `rewire_job_ads_analyser_server` folder with all its contents, the algorithm written in Java, the Web scraping in Python and the dictionaries are uploaded to the server.

The **next** necessary **step** to make the web application work is the part that secures the communication with the client. The principle of communication is shown in the Fig. A.1.

1. Client is sending a POST request with a URL parameter. URL can look like this: `https://www.linkedin.com/jobs/view/9999999999/` (green arrow).
2. Backend Web Server prepares those data to acceptable format using web scraper, dictionary search algorithm and PHP text (URL) formatting (purple box).
3. Server is sending HTTP 200 OK response with text data formatted in JSON structure (shown in Listing A.1), that is ready to be displayed in the web application (red arrow).

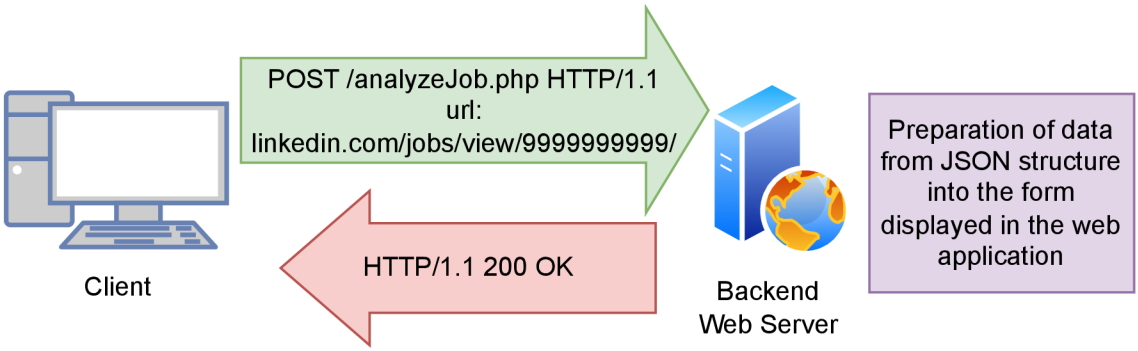

Fig. A.1: Principle of communication with web server

Listing A.1: Structure of HTTP response data

```
{
    "error": false,
    "message": "ok",
    "data": {
        "analyzed_results": {...},
        "job_description": {...},
        "new_form_data": {
            "title" => "Cybersecurity␣specialist",
            "country" => "Czechia",
```

```
            "company" => "AAA",
            ...
        }
    }
}
```

In this thesis, the "Client" folder is dedicated to this, containing code written in ReactJS language. Within this folder, running a web application in the localhost is also possible. To run this task, it is necessary to download NodeJs version 14.17.5.

To **install NodeJS** version 14.17.5, follow these steps:

```
$ sudo apt install curl
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/
install.sh | bash
$ source  /.bashrc
$ nvm install v14.17.5
```

Verify the correct version is installed: `$ node -v`

After a successful installation, open the terminal in the `opt/lampp/htdocs/client` folder and install the server using `$ npm install` command.

And that's the **installation** part **done**. The following command combination should be sufficient to start the server.

Start XAMPP:

```
$ sudo /opt/lampp/./manager-linux-x64.run
```

In the opt/lampp/htdocs/client folder, **launch the web application**:

```
$ npm start
```

Useful commands:

When MySQL server is down and wont start do this:

```
$ sudo service mysql stop
$ sudo service mysql start
```

Python script isn't working in web app, there might be problem with permissions use those commands:

```
$ sudo chown user:root opt/lampp/htdocs/rewire_job_ads_analyser_server/
sc_algorithm
$ sudo chown user:root opt/lampp/htdocs/rewire_job_ads_analyser_server/
sc_algorithm/main.py
```

A.2 Instructions For Work With Java Program

There are 2 folders in the appendencies. The first folder aho-corasick contains the main code with the program and the second folder called KeywordGen contains the

subprogram for creating keywords. For these codes to work, you need to have JAVA and MAVEN installed and have their paths set in System Path Variables. To make aho-corasick work, you need to add 3 dependencies in the pom.xml file and then load them using Maven.

```
<dependency>
  <groupId>org.ahocorasick</groupId>
  <artifactId>ahocorasick</artifactId>
  <version>0.6.3</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>5.2.2</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>5.2.2</version>
</dependency>
```

Similarly, for KeywordGen you need to add 2 dependencies in the pom.xml file and then load them using Maven.

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>5.2.2</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>5.2.2</version>
</dependency>
```

B Content Of The Electronic Attachment

- `/` ..... root directory of the attached archive
  - `bc_ahocorasick` ..... main program with sorting algorithm
    - `src`
      - `interval` ..... files allowing mapping of keyword positions
      - `trie` ..... files allowing to create trie, suffix and output links and matcher
      - `main.java` ..... the main file for starting the program
    - `.gitignore` ..... files that will be ignored for upload to git
    - `LICENSE.md` ..... the original license of the used program
    - `pom.xml` ..... configuration file
    - `README.md` ..... short description of the program
    - `dictionaryone100.xlsx` ..... dictionary
    - `dictionaryoneLow.xlsx` ..... second dictionary
  - `KeywordGen` ..... subprogram for input creation
    - `src`
      - `main.java` ..... the main file for starting the subprogram
    - `.gitignore` ..... files that will be ignored for upload to git
    - `pom.xml` ..... configuration file
    - `README.md` ..... short description of the subprogram
    - `dictionaryshort.xlsx` ..... combination of 2 dictionaries
  - `rewire_job_ads_analyzer_server` ..... files placed on the server
    - `ac_algorithm` ..... files needed for java program
      - `aho-corasick.jar` ..... executable jar file
      - `dictionaryone100.xlsx` ..... input dictionary
      - `dictionaryoneLow.xlsx` ..... input dictionary
    - `sc_algorithm` ..... files needed for python program
      - `chromedriver` ..... browser for opening by script
      - `main.py` ..... executable python file
    - `analyzeJob.php` ..... file that manages automation