Obraz zawierający tekst

Opis wygenerowany automatycznie

**Computer Science**

**Informatyka**

Data Science

**Szymon Wujec**

S20431

**“Data analysis in terms of resource usage based on “Bike Sharing” dataset”**

Master Thesis

Thesis supervisor

Dr. habil. Grzegorz Marcin Wójcik

Warsaw, ------------, 2024

Table of contents

[1 Introduction 4](#_Toc127208227)

[1.1 Thesis structure description 4](#_Toc127208228)

[1.2 Goals of the thesis 4](#_Toc127208229)

[1.3 Main thesis issues 4](#_Toc127208230)

[1.3.1 URL address 4](#_Toc127208231)

[1.3.2 Web aplication 4](#_Toc127208232)

[1.3.3 Bot 5](#_Toc127208233)

[1.3.4 Web Scraping 5](#_Toc127208234)

[1.3.5 Framework 5](#_Toc127208235)

[1.3.6 Refactoring 6](#_Toc127208236)

[1.3.7 Markup language 7](#_Toc127208237)

[2 Description of application. 11](#_Toc127208238)

[2.1 Description introduction 11](#_Toc127208239)

[2.2 Front-end description 11](#_Toc127208240)

[2.3 Back-end description 11](#_Toc127208241)

[2.4 Input and output of data file 12](#_Toc127208242)

[2.5 Tools 14](#_Toc127208243)

[2.5.1 IntelliJ IDEA 14](#_Toc127208244)

[2.5.2 Excel 16](#_Toc127208245)

[2.5.3 Python 16](#_Toc127208246)

[2.5.4 Selenium 18](#_Toc127208247)

[2.5.5 Pandas 19](#_Toc127208248)

[2.5.6 Openpyxl 19](#_Toc127208249)

[2.5.7 HTML 20](#_Toc127208250)

[2.5.8 Google Maps 21](#_Toc127208251)

[2.5.9 Materialize 23](#_Toc127208252)

[3 Functionality process assumptions 25](#_Toc127208253)

[4 Implementation. 30](#_Toc127208254)

[4.1 Front-end code 30](#_Toc127208255)

[4.1.1 Home page 30](#_Toc127208256)

[4.1.2 Summary page 34](#_Toc127208257)

[4.2 Back-end code 35](#_Toc127208258)

[4.2.1 Initialization classes 35](#_Toc127208259)

[4.2.2 Callable classes 41](#_Toc127208260)

[5 Summary 51](#_Toc127208261)

[5.1 Application outcome 51](#_Toc127208262)

[5.2 Acknowledgments 51](#_Toc127208263)

[6 Bibliography 52](#_Toc127208264)

[7 Attachment 53](#_Toc127208265)

[7.1 Table of Figures 53](#_Toc127208266)

[7.2 Table of Pictures 53](#_Toc127208267)

[7.3 Table of Code 54](#_Toc127208268)

[7.4 What CD contains 55](#_Toc127208269)

# Introduction

## Thesis structure description

The main goal of the work was to create a tool that will help employees working in corporations, better manage and reduce time-consuming tasks that contribute nothing to the employee's own development. The idea for such a web application was based on my personal experience and lessons learned from the tasks assigned to me in the company where I worked. This corporation offers a wide range of services in the context of customer portfolio management, which includes large companies as well as small start-ups. The client's portfolio consists of its ownership, and the subject of work is based on geolocation surfaces. The competition from which the client passed did not collect the required and key information for us, the task I was given was to search the Internet based on Google Maps search results and fill in the missing data.

## Goals of the thesis

The application "GeoFinderUI" using the available databases on the Internet will supplement with suggestive data, selective data sent to the application by means of compilation of minimal information on a specific location in a spreadsheet. This is intended to reduce the user's involvement in searching the results on the Internet and to streamline the work associated with tasks that require the collection of geolocation data of a specific location.

## Main thesis issues

### URL address

A URL is a Uniform Resource Locator, most commonly associated with the World Wide Web (WWW) is nothing more than a standardized format for addressing information, applications or data on websites or the local network. A user, after typing a specific existing URL in his browser, is redirected to a page with an assigned, user-specified address.

(Uniform Resource Locator, 2022)

### Web aplication

A web application is a program running on an external or local server with the ability to connect, communicate with and use it over a computer network with the user's host in his private session. For this purpose, compatible devices having a web browser are used, such as a personal computer, laptop, smartphone, smartwatch, smart TV and many other such devices.

(Aplikacja internetowa, 2022)

### Bot

A bot is a program that performs a predetermined action instead of a human, and in many cases has the task of imitating human behaviour that does not go beyond a defined action. The word Bot comes from the word robot which means automated machine. In these days of increased technological development, we encounter such machines at every step, in the kitchen, on the way to work, school and even in stores. A popular place for the use of bots are community forums, where bots have the task, in the absence of staff, to supervise the content emoted by users so that it complies with the rules of the portal. Unfortunately, not everywhere bots are assigned the task of policing the rules, for example FPS or TPS games where bots, known as “aimbots”, assist players in aiming and automatically set the sight themselves. This is against the rules of these types of games, which often means that you must reckon with an account ban imposed by an administrator for using boosters.

(Bot program, 2022)

### Web Scraping

Web scraping is a process used in the Internet environment, performed by bots that have been created by a programmer to automatically retrieve data appearing on a computer network, the World Wide Web. Unlike the process called screen scraping, which only copies the pixels displayed on the monitor screen, web scraping reads the entire HTML source code of a particular web page. Then, using keywords, page location or the type of information posted, such as the header or page title, the bot collects information of interest to the author of the code. Unfortunately, many sites defend themselves against facilitated and massive data collection, so many websites automatically detect and block sessions under the pretext of using bots.

(Web scraping, 2023)

### Framework

A framework is a skeleton for a web application. Each framework has a set of components and available libraries adapted to program the application. There are many frameworks available, which differ from each other, so you can choose the right development platform for your application depending on the required structure or general mechanism of program operation. The advantages of using frameworks in programming are efficiency, improved code quality and reliability. Frameworks, by having pre-defined and pre-written tools, require less code to be written by a programmer using a particular development platform, which translates into efficiency. The same applies to improving the quality of the code, the predetermined structure of the written code is logically and thoughtfully created with a view to the flexibility of the program and at the same time affects the quality of the written content. Reliability, on the other hand, is achieved by a logical and well thought-out writing of the framework, which has undergone many complex tests before being released in an official version for public use. Areas of application of frameworks are: compilers of various programming languages, financial modelling applications, earth science modelling applications, systems, decision support, multimedia frameworks, web applications, middleware, web application frameworks.

(Framework, 2022)

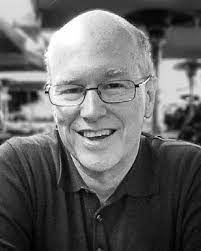
### Refactoring

Refactoring is the name of a technique involving the restructuring of an existing body of code, interfering with its structure, without changing the operation and functionality of the script. This procedure is very popular among programmers in order not to duplicate repeatedly occurring parts of the code, but to call a specific one method using variables, thus making it universal. As a result, the script takes the form of a clear, readable, and transparent code, which takes up much less code than before the refactoring procedure. The first known use of the term refactoring took place in 1990 in the published literature. The authors of the article were William Opdyke and Ralph Johnson. Martin Flower is a British software developer. His experience in the industry is passed on to the next generation in many public statements, publications, and books. One of those books is a book titled "Refactoring: Improving the design of Existing Code," published in 2002 in collaboration with the aforementioned William Opdyke.

(Code Refactoring, 2023)

“Help in understanding the code also helps me spot bugs. I admit I'm not terribly good at finding bugs. Some people can read a lump of code and see bugs, I cannot. However, I find that if I refactor code, I work deeply on understanding what the code does, and I put that new understanding right back into the code. By clarifying the structure of the program, I clarify certain assumptions I've made, to the point at which even I can't avoid spotting the bugs. 49 It reminds me of a statement Kent Beck often makes about himself, "I'm not a great programmer; I'm just a good programmer with great habits." Refactoring helps me be much more effective at writing robust code.”

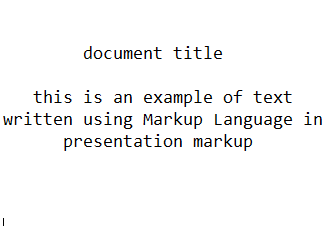
(Flower, Beck, Opdyke, & Brant, 2002)

**Picture 1 Left side, Martin Flower Picture 2 Right side, William Opdyke**

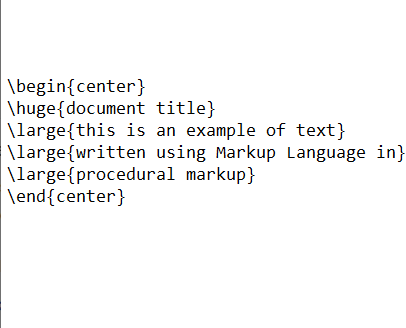
### Markup language

Markup language is nothing more than a form of code notation made up of text and tags representing a particular element in the code. An example of the most popular markup language is "HTML (Hyper Text Markup Language)", this technology is the basis of "World Wide Web (WWW)" sites. "HTML" has so dominated the market of similar technologies that when talking about code that is the result of the combination of text and markup, the first thought of the recipient is likely to be "Hyper Text Markup Language", while it is not the only language in this category, text formatting technologies. In fact, there are many such languages, we can meet them as applied technologies in environments such as business, economy, finance, culture, media, entertainment, science, technology, engineering, and mathematics. The first Markup Languages were used, and are still used today, in the paper, editorial and publishing industries. The many languages available, adapted to different subject environments, result in the division of markup languages into 3 groups of these languages. Presentational markup, procedural markup and descriptive markup. Presentational markup does not require any special formatting of the document, the only characteristic is the structure of the stored information. For example, when using presentational markup, in order to mark the margins and to achieve the effect of centering the text, several characters are used to start a new line or a space character, which precede the text information value located in the line of the document.



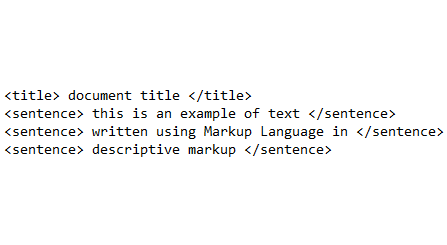
**Figure 1 Presentation markup example**

Procedural markup, like the previous example, is responsible for presenting textual information value. The difference between procedural markup and presentation markup, however, is that the markers are no longer space characters or the start of a new line, but some kind of command to the program or device reading the document so that when displaying the text contained in the document, the end user does not see the markers used by the author of the original document record.



**Figure 2 Procedural markup example**

The example of formatting text in procedural markup, shown above, was written using the "TeX" language. The last type of text formatting using markup language is descriptive markup. This type, unlike the previous two varieties mentioned, does not refer directly to the formatting of the layout of the text on the page of the document, but with the help of appropriate marking of the separated phrases, the author is able to determine what it means and what role in the whole document plays the extracted text fragment.



**Figure 3 Descriptive markup example**

Many of the languages in the category of languages created for formatting text and transmitting data in this way, use in their structure data notation from a combination of the above-mentioned varieties of markup languages. An example of such a language is "HTML" in which descriptive markup is predominant.

(Markup language, 2023)

# Description of application.

## Description introduction

The application "GeoFinderUI" is a hosted tool, for example, using a local or external server. Access to it is through the user's browser. This makes it a web application. After entering the correct URL, the user is redirected to the application's web start page.

## Front-end description

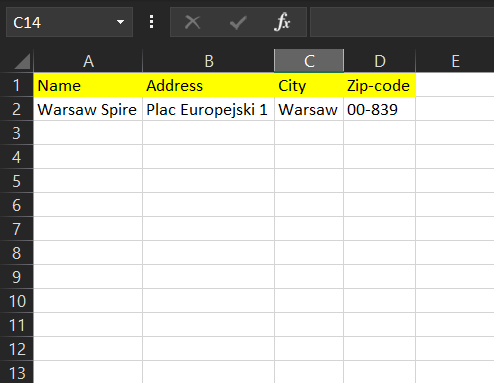
Starting from the top of the page, there is a "Download file here" button. This is the button that initiates the procedure for downloading the file. In this case it is the file "Template.xlsx". This is an example format for entering data using Microsoft Excel spreadsheets from the Microsoft Office suite. The example file contains items such as the name of the place, the address of the location being searched for, the city and the postal code. As an example, I will describe the location of our university, "Polish-Japanese Academy of Computer Technology". The values of the cells forming a full row with data are presented as follows Name: PJATK, address: Koszykowa 86, city: Warsaw, postal code: 02-008. Below there is a button that opens a file explorer window with the content "Choose File" in order to select the file "Template.xlsx" filled with data. After finding the location where the file is stored on the user's local disk, and confirming your choice, the "Proceed the file" button should be clicked to start the main application process. At the very bottom of the page there is a "drop-box" button, which allows the user to decide how much delay in seconds the program should pause when automatically searching for phrases on web pages. This is necessary to avoid or reduce the number of occurrences in the received data of a value equal to "N/A", meaning that the program will not find the information searched for. This is due to the different stability and speed of the Internet connection. In this case, if the user is aware of the poor quality of his/her Internet connection, he/she should choose one of the numbers from the recommended range of 1 to 3 seconds of delay, where the higher the number, the greater the chance of finding the searched value. It should also be borne in mind that the larger the value you set, the more the time to receive the information you are looking for will increase.

## Back-end description

The user, after attaching all the necessary files and setting the parameters suitable for him, approving them and starting the main process of the application through the button "Proceed the file", is left in the stage of loading the web page. During this time, the program written in the "Python" language with the file name containing the main script "home.py", retrieves and analyses all the information provided by the user in order to eventually return to the user a file filled with the suggestion data prepared by the program. In this process, the first step is to check whether the running method has a "POST" value. When this condition is satisfied, the next condition is whether the user has transferred the spreadsheet file with the data before running the procedure. Then the program proceeds to check the folder stored in the files of the "GeoFinderUI" application, which in subsequent stages will be used to transfer, save and store the "Excel" file with the basic name "Template.xlsx" containing the data provided by the user. If the program encounters any files there, they will be deleted. Such a phenomenon occurs when the program is not used for the first time and no one has cleared this folder before. The next step is to read the uploaded files and assign them to variables for easier handling in subsequent steps. If the user has specified the optional value mentioned in the previous section, concerning the delay of information retrieval, this parameter is also retrieved. Then the program opens the spreadsheet file and checks the correctness of the data format. This process reads the column headers and then checks that the correct data is in the correct columns. When this condition is met, using the "for" loop, each row is read in turn, and the acquired information is assigned to the variables that the program will use when searching for phrases on the Internet. The automated browser "Google Chrome" is started in stealth mode and the whole process of searching for data is carried out as a background process. When the program obtains the searched information, it is entered into the spreadsheet as suggestive data. After the program has collected all possible information, the user is taken to a summary page. When the program detects an error, the page displaying the summary will contain information for the user where the error occurred. If the process is executed correctly, on the final page the user is shown a button initiating the download of a spreadsheet prepared by the program, with a new name "S\_DataValues.xlsx".

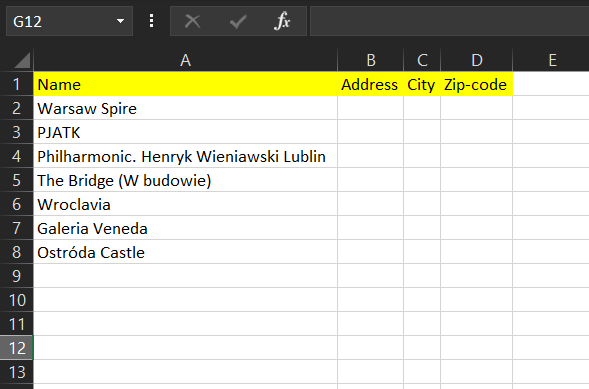
## Input and output of data file

The basic template file downloaded from the application page is a file very truncated in the input data. The input data is intended to tell the user what information the program expects in specific columns of the spreadsheet.



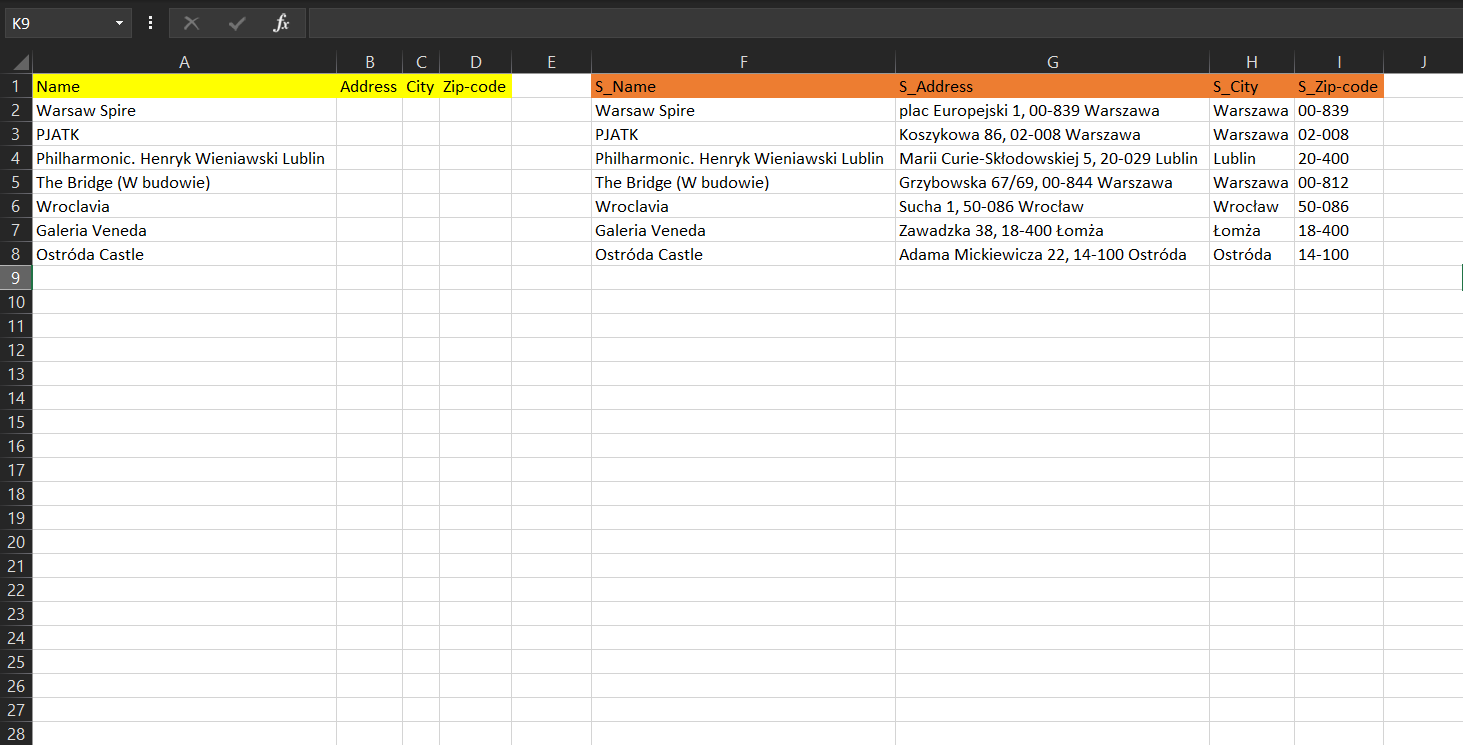
**Figure 4 File Template.xlsx**

For testing purposes, a separate file was prepared having more input data regarding other locations. The information provided is the names of specific locations, on the basis of which the application is to prepare a full set of suggestion data.



**Figure 5 Input file with test data**

The result of the input of the above information and its processing by the "GeoFinderUI" application is the file "S\_DataValues.xlsx". The application has completed the process of acquiring missing data on the basis of the provided minimal input data with the result of 100% efficiency and correctness of the data.

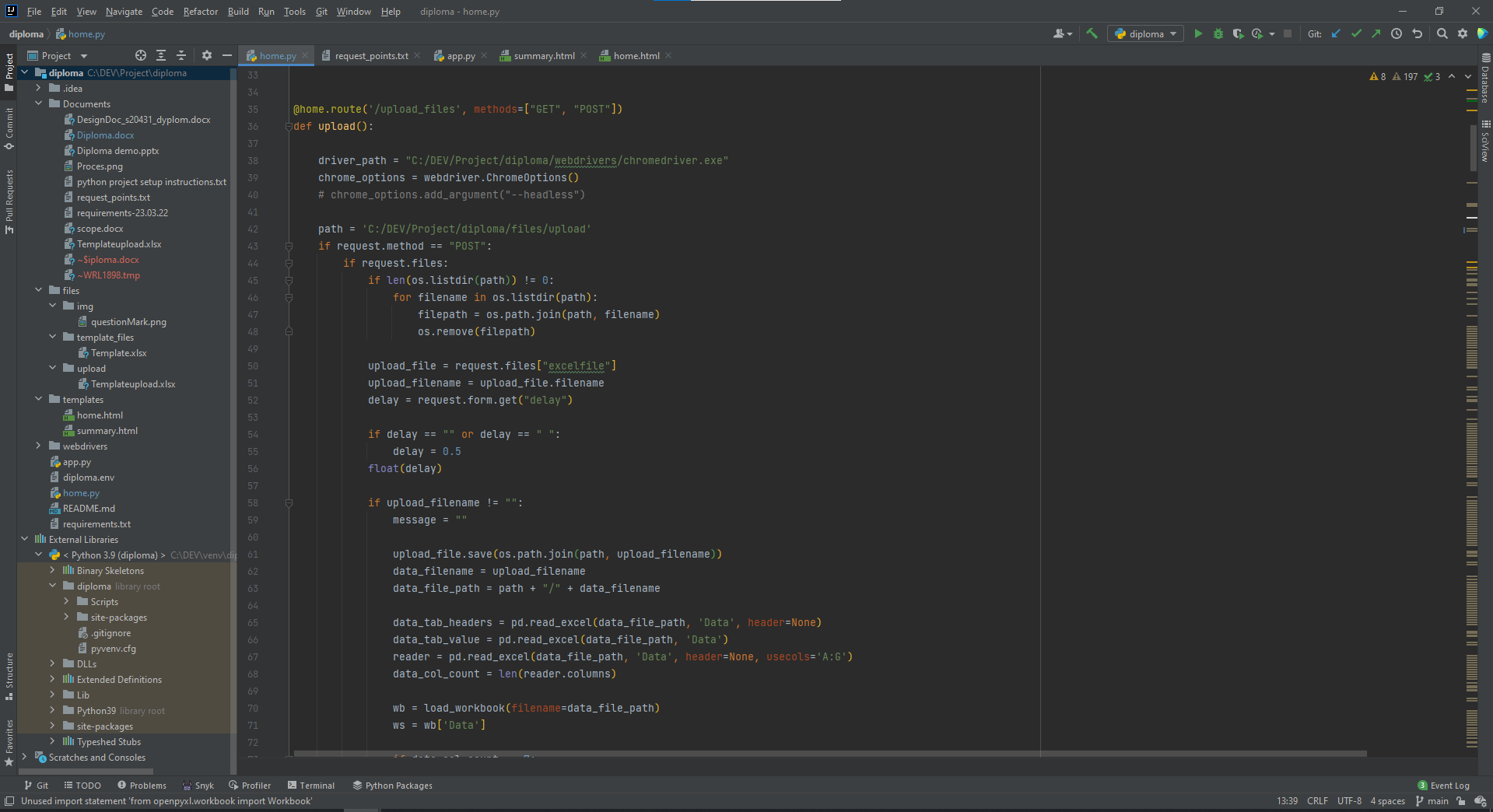
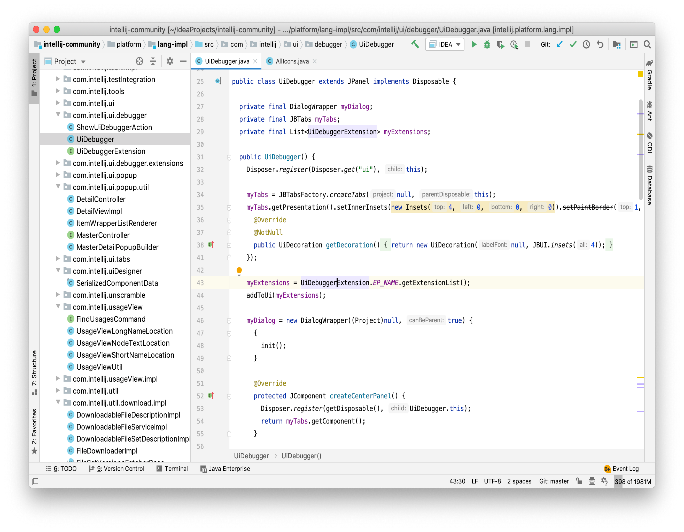


**Figure 6 Output file with test data**

## Tools

### IntelliJ IDEA

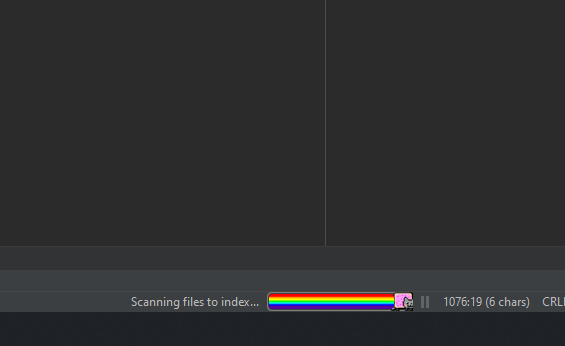
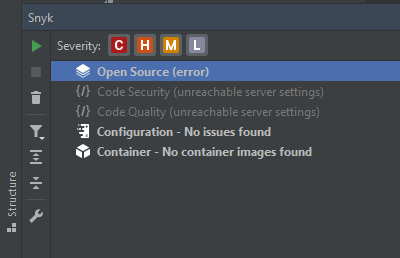
In order to create advanced or those simple, recreational IT programs for which the programmer or the organization for which the programmer is writing the program, a code editor is needed. A great editor is IntelliJ IDEA, a commercial integrated development environment that can compile a written script. The editor was written in Java language by the company "JetBrains". The first publication appeared on the market in the first quarter of 2001. This version had a set of tools to support refactoring. Currently, it offers many helpful tips and keyboard shortcuts to assist the programmer in his work. It supports 19 programming and automation languages such as "Java", "Python", "Scala" or languages designed for websites like "HTML", "CSS". IntelliJ IDEA also provides interoperability with many other open source development environments or tools, such as "GIT", "Apache Maven", "Apache Ant", "JUnit", "CVS" or "SVN". The editor also has an interesting for many IT fanatics, dark theme of the interface layout.

**Picture 3 Left side, IntelliJ dark theme Picture 4 Right side, IntelliJ default light theme**

There are many plugins available to extend the functionality of the raw version of IntelliJ IDEA such as "Rainbow Brackets" by "izhangzhihao" which improves code readability by cascading colors of overlapped rounded, square and bracket brackets, or the plugin "Snyk Security - Code, Open Source, Container, IaC Configurations" which analyzes the ko di written by the programmer and returns a report on the security of the created application. The reports are presented in a very readable manner and pose no problem in understanding them for novice programmers. "Snyk" also presents potentially possible solutions to problems related to security vulnerabilities. There are also recreational plugins created to bring a smile to the face of the IntelliJ IDEA code editor user, one such plugin is "Nyan Progress Bar" by "Dimitry Batkovich", which turns the existing compiler loading bar into a rainbow tail of the iconic character in the development community that is "Nyan Cat".

(Discover Intellij Idea, 2023)

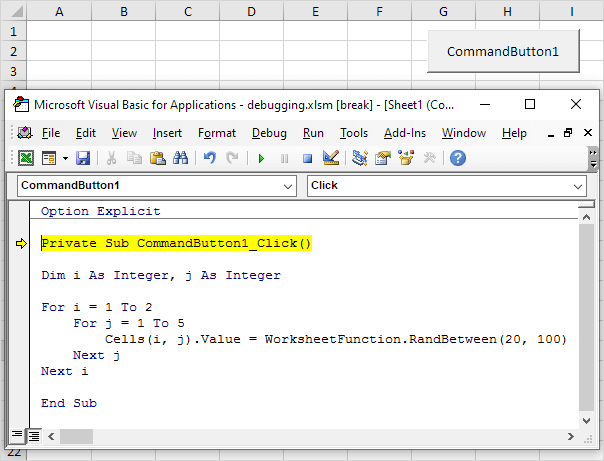
 

**Picture 5 Left side, Nyan Progress Bar Picture 6 Right side, Snyk add-in**

### Excel

Microsoft Excel is a spreadsheet created and still being developed for use on Windows, macOS, Android and iOS operating systems. With its help the user is able to easily calculate various mathematical operations, present them in a graph or pivot table. You can even write a program yourself to automate the process required by the user called Macro. The language needed to write such a Macro is Visual Basic For Applications (VBA). With each newly released version of this program we get many new features and improvements, as well as simplifications for the user. A lot of additional tools that are not provided by the official developer release can be found in the form of add-ins, mostly they are third-party software, most often written by private developers, these are plug-ins that extend the functioning of Microsoft Excel.

(Microsoft Excel, 2023)

**Picture 7 Left side, Excel Worksheet Picture 8 Right side, Visual Basic for Applications**

### Python

A programming language with many frameworks dedicated to specific tasks, applications or programs and with a very extensive list of standard libraries is a wide range of possibilities for the programmer. The main idea behind this language is the clarity and readability of the source code. The main feature of the syntax is clarity and conciseness. Python supports various types of programming such as imperative, object-oriented and, to a much lesser extent than the previous two, functional. A distinctive point is the dynamic type system and automated memory management, which is why you won't encounter variable declarations in it, for example, integer, double, String. Python is a very popular programming language, which makes its interpreters available for many operating systems. The development is carried out as open source, whose trustee is the "Python Software Foundation", modelled on the "Apache Software Foundation". It is a non-profit organization. The implementation of the language, which is also standard, is written in the "C" language, "CPython". It is standard because others also exist. The implementation written in the "Java" language is JPython, and written in "Common Lisp" and "IronPython" is "CLPython", for the ".NET" platform. The beginnings of Python took place in the early 1990s, its creator was Guido van Rossum. The language was intended to be a successor to the "ABC" language, and was developed at the Center for Mathematics and Computer Science in Amsterdam (CWI - Centrum voor Wiskunde en Informatica).

**Picture 9 Left Side, CWI building Picture 10 Right side, Guido van Rossum**

Interestingly, the designer did not refer to an animal when he proceeded to choose the name of the programming language. He was a big fan of the comedy series, popular in the 1970s aired on "BBC" television under the title "Monty Python's Flying Circus". Based on information gleaned from the Internet, in 2020 there were 25 frameworks available for public use, dedicated to the Python language, written in Python as well as those written based on it. The "GeoFinderUI" application was written using the "Flask" framework. Flask is a micro web framework, it has this backend because it does not require individual tools or libraries. It is a great choice when designing and programming a web application.

(Python, 2023)

### Selenium

Selenium is an open source project, most commonly found in the form of a library available for many programming languages. The purpose of this package is to enable easy automation of access to the browser. A possible use of the functionality of this powerful tool is, for example, to search for a specific phrase typed automatically by the program, and then using Web Scraping technology, to find the information of interest to the user stored in the source code of a web page written in "HTML". For this to be possible, in addition to a correctly stored automation routine in a script of one of the languages including "C#", "JavaScript (Node JS)", "Groovy", "Ruby", "Java", "Perl", "PHP", "Scala" and "Python", a browser driver is needed. The official Selenium website provides 5 drivers for the most popular browsers, "Chrome", "Firefox", "Edge", "Internet Explorer" and "Safari". The package is available for 3 operating systems "Windows", "Linux", "macOS". The author of Selenium is Jason Huggins. In 2004 he created the tool for internal use in the company "ThoughtWorks". The project was quickly recognized as a great achievement and received great publicity. In 2005, Dan Fabulich and Nelson Sproul proposed the release of a series of patches to transform what was known as "Selenium Remote Control (RC)" into the Selenium tool very well known today in the IT community using mainly Web Scraping technologies. The name Selenium comes from a joke contained in one of the emails sent by Huggins. The joke was intended to foreground the competition, which was Mercury Interactive Corporation. Mercury is an Israeli company bought out by global technology giant HP Software Division. The humorous email had in its subject line the phrase "you can cure mercury poisoning by taking selenium supplements".

(Selenium, 2022)



**Picture 11 Jason Huggins**

### Pandas

Pandas is a library written for use in the Python programming language. It offers many possibilities for analysis and manipulation of data, mainly numeric data. This software is distributed in the form of a free subscription plan having a "BSD" license. The first publication took place in 2008 by Wes McKinney, who at that time worked for the company "AQR Capital Management". The original idea of constructing this tool was the need for a high-performance way to analyse financial data. The name of this library comes from the two-letter phrase "panel data", where the first letters form the word "panda". "Panel data" is an econometric term for data sets having observations over multiple time intervals.

(Pandas software, 2023)



**Picture 12 Wes McKinney**

### Openpyxl

When the program or application that the programmer designs requires the use of "Excel" spreadsheets from the "Microsoft Office" package, the best solution for automated handling of this program is the "Openpyxl" library. "Openpyxl "is a well-developed library containing many useful spreadsheet control tools. The main reason for the creation of this package was the lack of existence of similar solutions capable of opening and editing files saved in formats with extensions "xls", "xlsx", "xlsm", "xltx and "xltm" in a simple, clear and logical way. Unfortunately, in the world of computer technology there are many threats in the form of viruses or spyware, which can often be encountered very well hidden in Excel spreadsheets coming from an unknown source. According to the official website of the manufacturer, the library "Openpyxl" does not offer any support in this regard. The only possible option suggested by the owners of the software is to install an additional package that is "defusedxml". The authors of the "Openpyxl" software are Eric Gazoni and Charlie Clark. The first version available for public use was version 1.7.0 released in late 2013.

(Openpyxl, 2022)

**Picture 13 Left side, Eric Gazoni Picture 14 Right side, Charlie Clark**

### HTML

The language "Hyper Text Markup Language (HTML)" belongs to the category of markup languages. The predominant variety of text formatting occurring in the "HTML" language is descriptive markup. Distinguish these varieties of markup of text fragments or transmitted data, which define the role of fulfilling the extracted information stored in the document. "HTML" was created for the creation of web pages "World Wide Web (WWW)", it allows to describe the structure of the information displayed inside a web page. The main tools available are the formulation of hyperlinks, lists, tables, headings, paragraphs and the possibility of embedding multimedia or file objects in the text of the document. There are also non-database elements, such as interactive data forms. The beginning of the language "HTML" can be marked in 1980, when physicist Tim Berners-Lee created a prototype of a hypertext information system called "ENQUIRE". The purpose of this system was to organize and make available documents related to scientific research. Users could then access the shared documentation located in a completely different, remote location. In 1989 Tim Berners-Lee competed with Robert Cailliau. The object of their competition was a parallel proposal for hypertext information systems based on the Internet. This was the beginning of an adventure leading to the development of an amazing technology, which today's Internet users know very well, because in 1990 Tim and Robert joined forces and presented a jointly prepared proposal for the "World Wide Web" project. They achieved success as their proposal was accepted by CERN. The first publicly available documentation of the language "HTML," then called "HTML Tags," was issued in 1991. The original structure of this language contained 22 tags adapted to mark the elements contained in a document. "HTML", although it was written on the basis of the "SGML" language, did not have in it a formal definition of the original markup language for easy storage of text documents with the possibility of simple transfer, display and printing in various electronic data transmission systems.

(HTML, 2023)

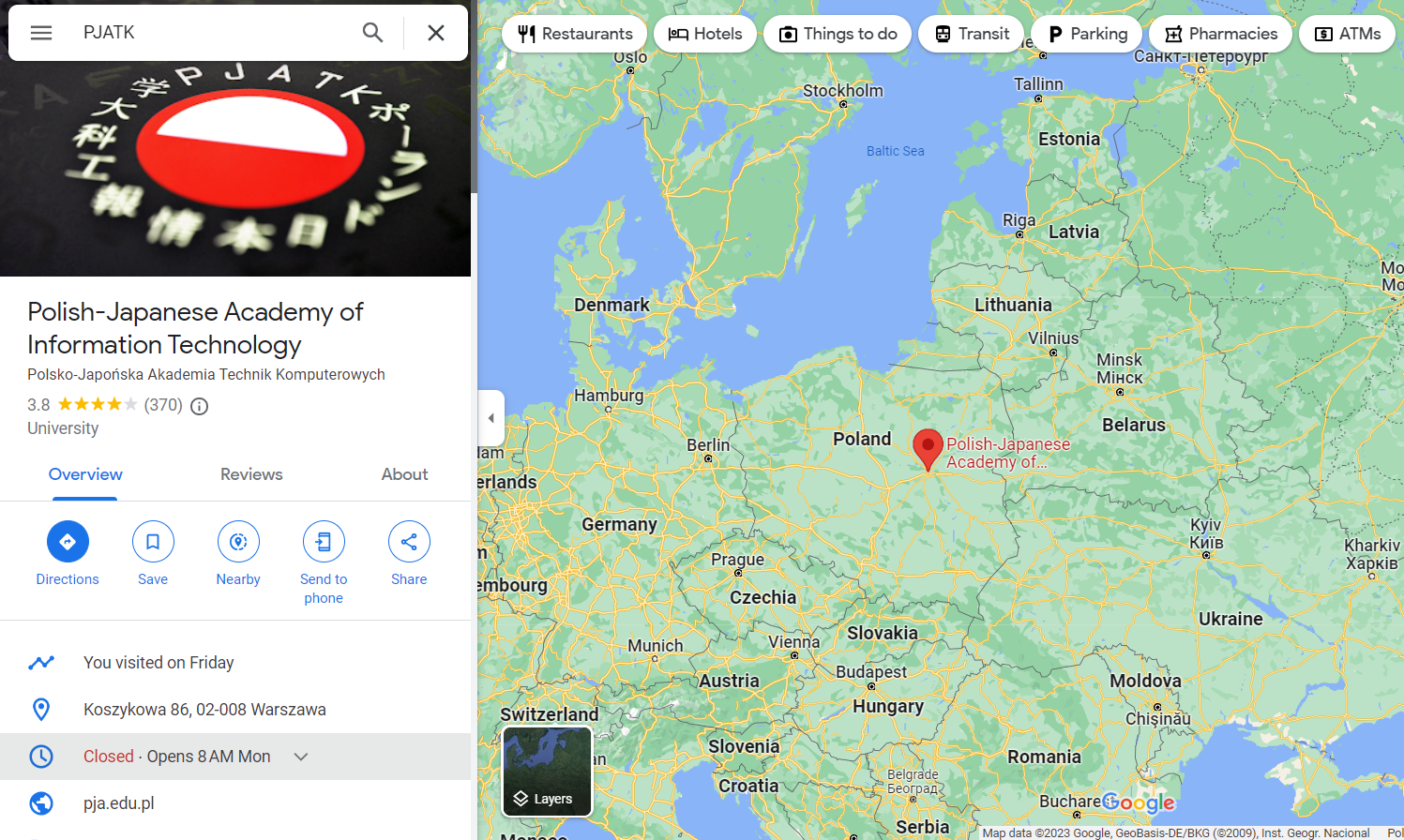
 

**Picture 15 Left side, Timothy Berners-Lee Picture 16 Right side, Robert Cailliau**

### Google Maps

The website, necessary for carrying out the main functionality of the application "GeoFinderUI" is "Google Maps". The company "Google", which owns the website in question, has a huge amount of data stored in databases, new items of which are added every second. The application "Google Maps" uses this data, which has in it information about the location located on the globe, and then presents it graphically in the form of marked points on the world map. In order to make it easier for the user to find a location of interest, the "Google Maps" application has been equipped with a search engine. After typing a phrase and finding it, the user is shown a list of the best matching search results. Unfortunately, such a search is not always consistent with the user's intentions. For example, the database on which the application operates is so large that there is a very high probability of finding two or even several locations having the same name. In order to identify the correctness of the search results, it is necessary to have a person or a program with analytical skills. Such scenarios do not occur every time, but every day there are many new business establishments, companies, green parks, streets, entertainment or tourist facilities, whose owners want their location to be available in the databases of the company "Google". This is the main problem that the user may encounter while using the application. Against this phenomenon, "Google Maps" is in constant development and improvement so that the user can get the best quality result of his search. However, the development is not only limited to the search terms entered by the user and the delivery of the results, the developers of "Google" are constantly creating new functionalities, which are made available to the public. Since the first version of this application, available to the public, released in 2005, many things have changed, for example, a very useful function "Street View" has been added, allowing the user to appear in the center of a designated location, available for this function, and view this place in the form of a virtual reality created with the help of interlocked images from a camera capable of taking so-called "360 photos." "360 photos" are combined projections acquired from at least two camera lenses, resulting in an expanded angle of view of up to 360°. Such images are usually saved in special formats that allow reading and displaying the file in computer programs dedicated to this technology and giving the user the opportunity to observe the selected frame, rather than the frame imposed by the author of the photograph.

(Google Maps, 2022)

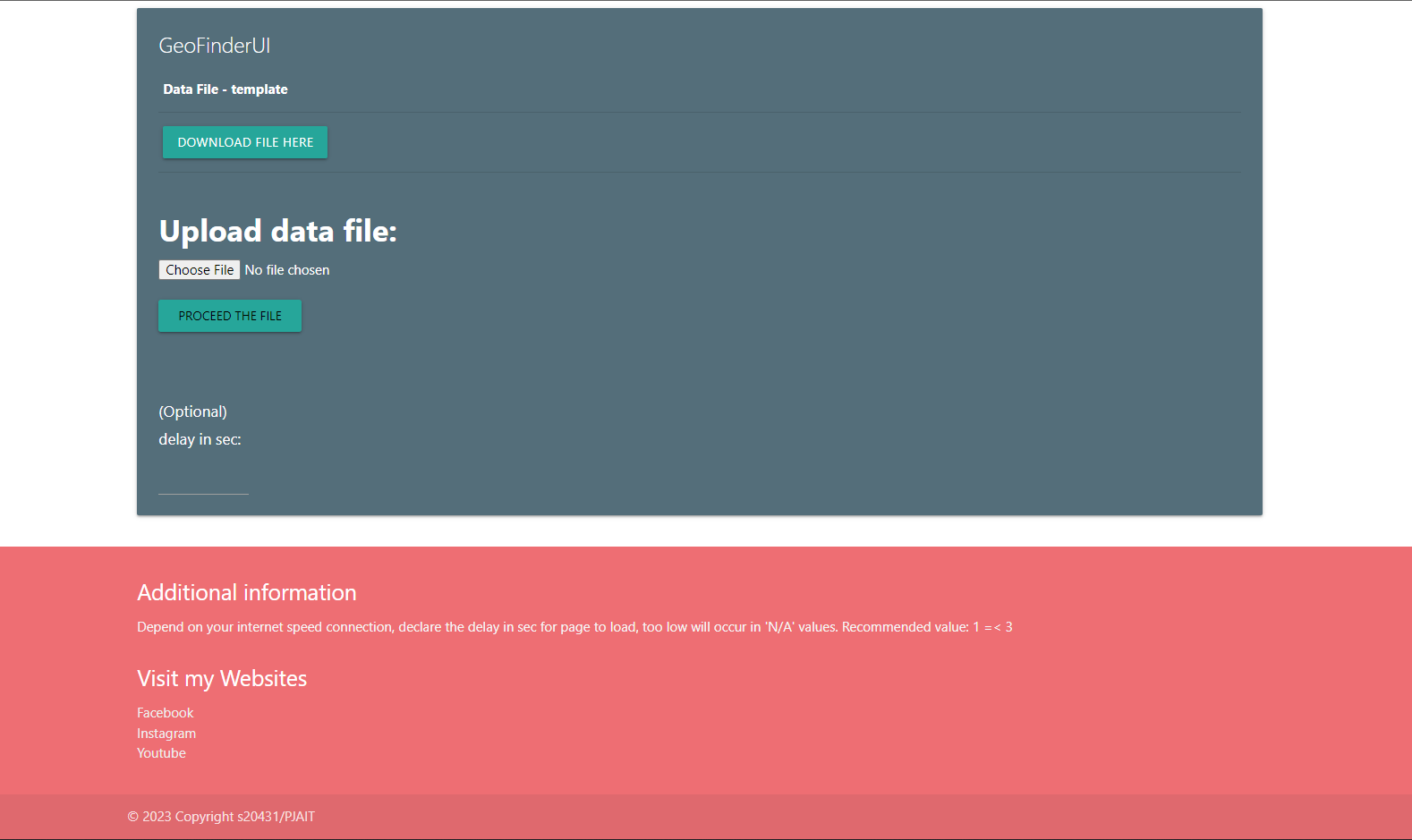


**Figure 7 Google Maps search result**

### Materialize

In order to better present your site and save a huge amount of time in the process, it is worth using ready-made "CSS" and "JavaScript" style packages prepared by the portal "materializecss.com". "Materialize" is a website that offers many categories of style packages for "HTML" pages. On the start page, the user is informed about the steps required before taking advantage of the benefits of building your own website and giving it a visual appearance through specially prepared tags of the "Materialize" portal. Each step is presented in a precise and logical way so that anyone would be able to understand and use this website. In order for the style packages to be available for use in the "HTML" document, you must add in the "<head> </head>"section two lines of text containing links to the page where the style packages are placed. The first is a link to the "CSS" packages, and the second is a link to the "JavaScript" packages. From the left side of the "Materialize" portal, the user can search for the package he is interested in from among many categories. In each category there are many examples with different appearance of the style package for a particular object. After finding the package of interest, it is necessary to copy the "HTML" code visible to the user under the example of the appearance of the styling of the "HTML" object, then place it in your document responsible for the layout of the web page and the content therein, written in "Hyper Text Markup Language".

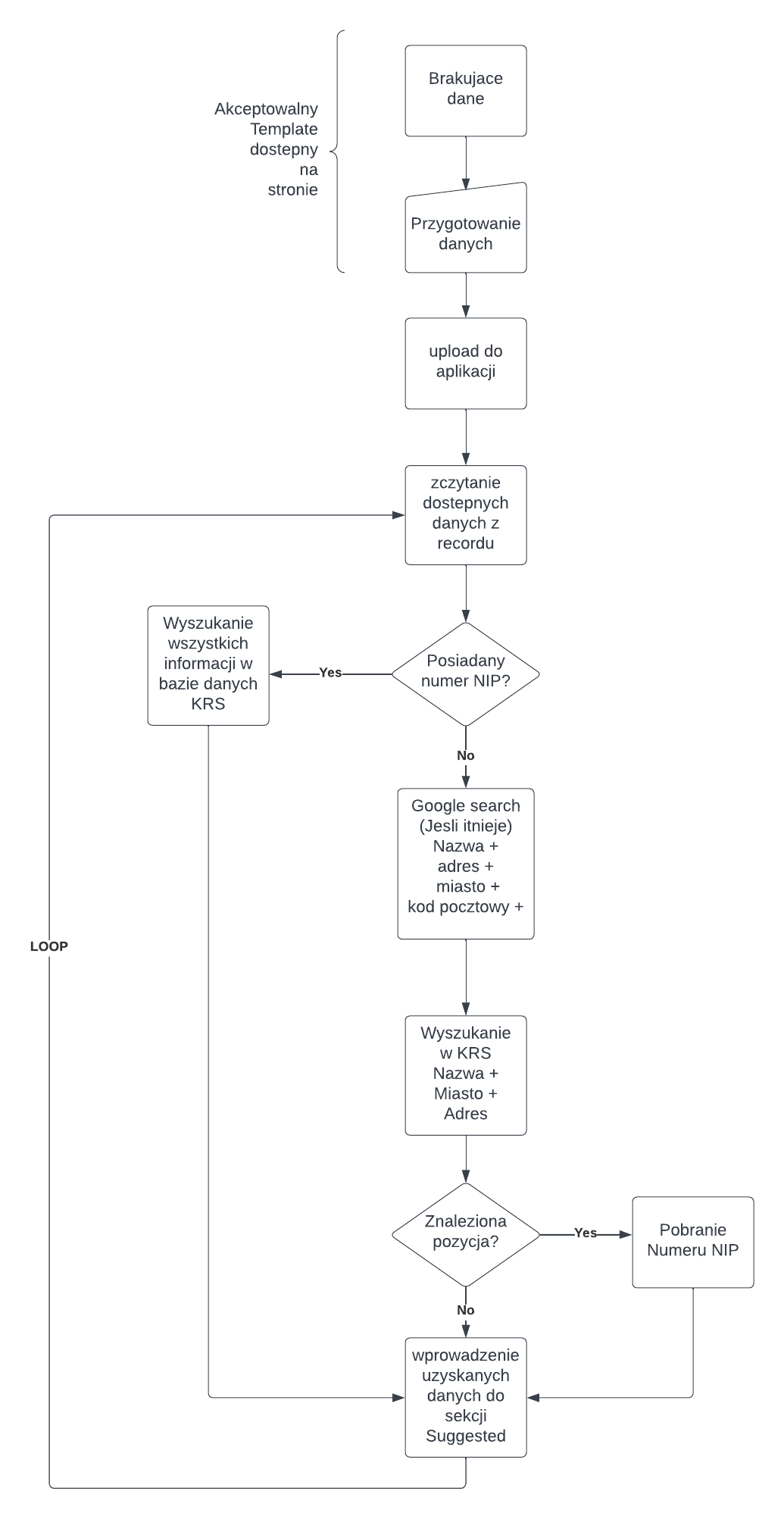
(Materialize, 2023)



**Figure 8 GeoFinderUI application home page with Materialized styling packages**

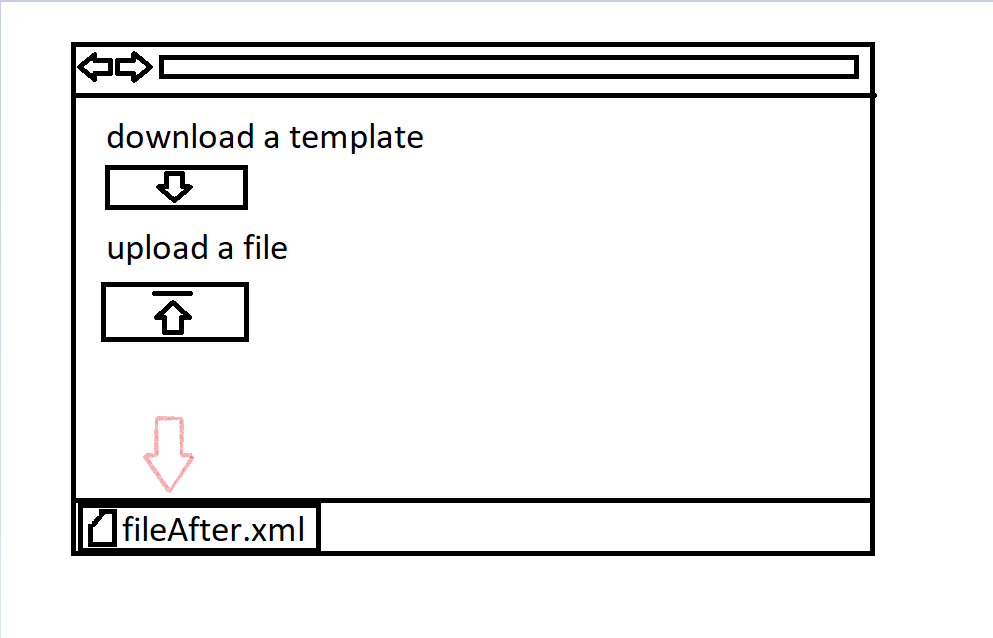
# Functionality process assumptions

The first stage of this work was the graphical formulation of the web application process. This process was mainly based on the entered "NIP" number or its search for a specific location. Such information is generally available on the National Court Register website "(KRS)". Unfortunately, these assumptions also included the use of web scraping technology. Without exception, the "KRS" website blocks access to the data on it after detecting automated movement through the site using a bot. Web scraping technology is the creation of such a bot capable of reading and searching the source code of a website.

****

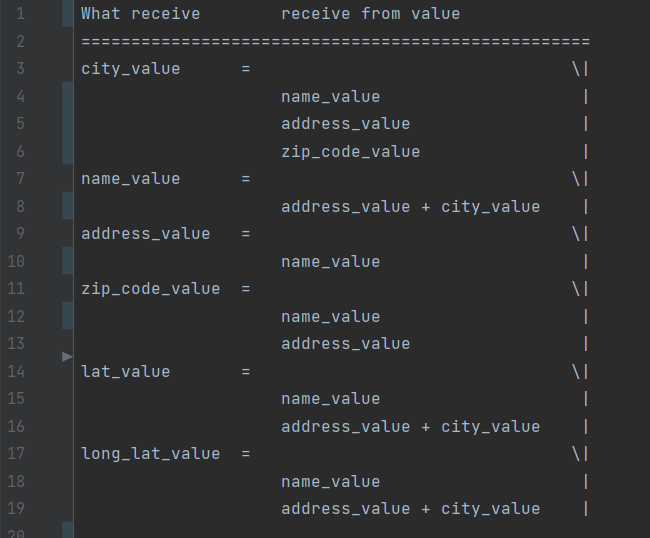
**Figure 9 the original diagrammatic process of the operation of the GeofinderUI application**

Such an obstacle required a restructuring of the functioning of the application. Since then, the main source of information acquisition became a portal having access to the largest geolocation database, which is "Google Maps". Before starting work on the programming of the application "GeoFinderUI", it was necessary to find the right tools that would help in obtaining the full functionality of the program and provide the best quality result to the user. The first requirements were to store and provide data for palliation in an understandable and simple way for the user. For this purpose, the choice fell on the spreadsheet "Excel". Due to the decision to use ".xls" files, the program, which was to be written in the programming language "Python", required appropriate libraries capable of handling spreadsheets. Library "Pandas" and library "Openpyxl" are well suited for such tasks. With their help "GeoFinderUI" reads and writes found data from the Internet in spreadsheets. The next step was to select a suitable tool for operating websites. The tool should make it possible to enter phrases into the search window of a browser or a web page, and then, among the search results, find an object in the source code of a web page, containing the searched information. The "Selenium" library meets the above requirements and is the most important part of the functionality of the "GeoFinderUI" program. The web application is not just a back-end script, for this we needed to visualize the layout of the page. A simple but succinct document, outlining the purpose to which the application was to aspire from the beginning, was prepared in the program "Paint" available in the operating system "Windows". This program is an easy to use and very limited in its capabilities graphical editor. However, it was sufficient to create a sample visualization of the appearance of the site.



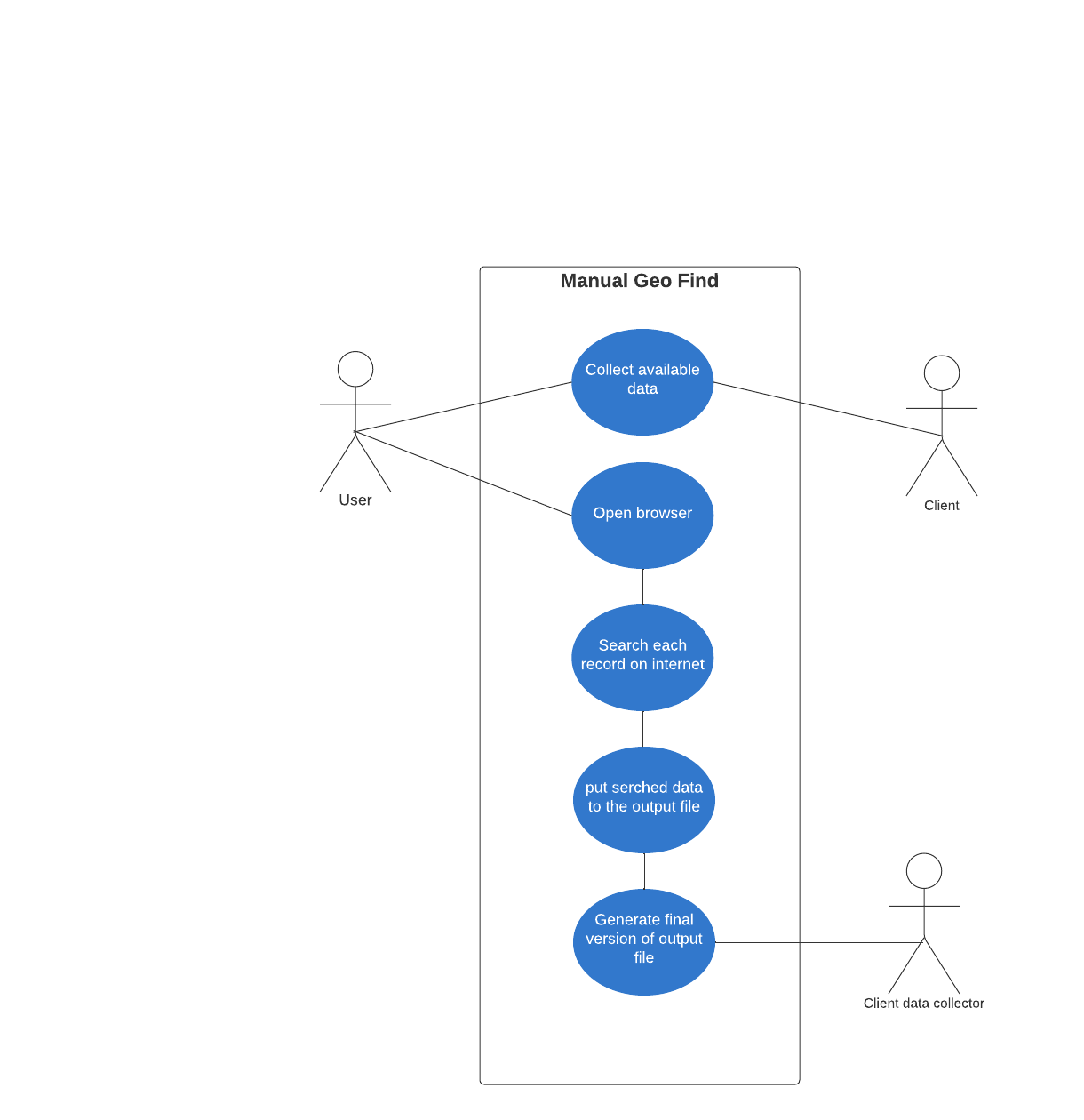
**Figure 10 sample visualization of the GeoFinderUI start page**

The next necessary step was to determine the possibility of receiving data. The purpose of this was to maximize the use of user-submitted data in order to receive the most available information about a given location in the final results prepared by the "GeoFinderUI" application.



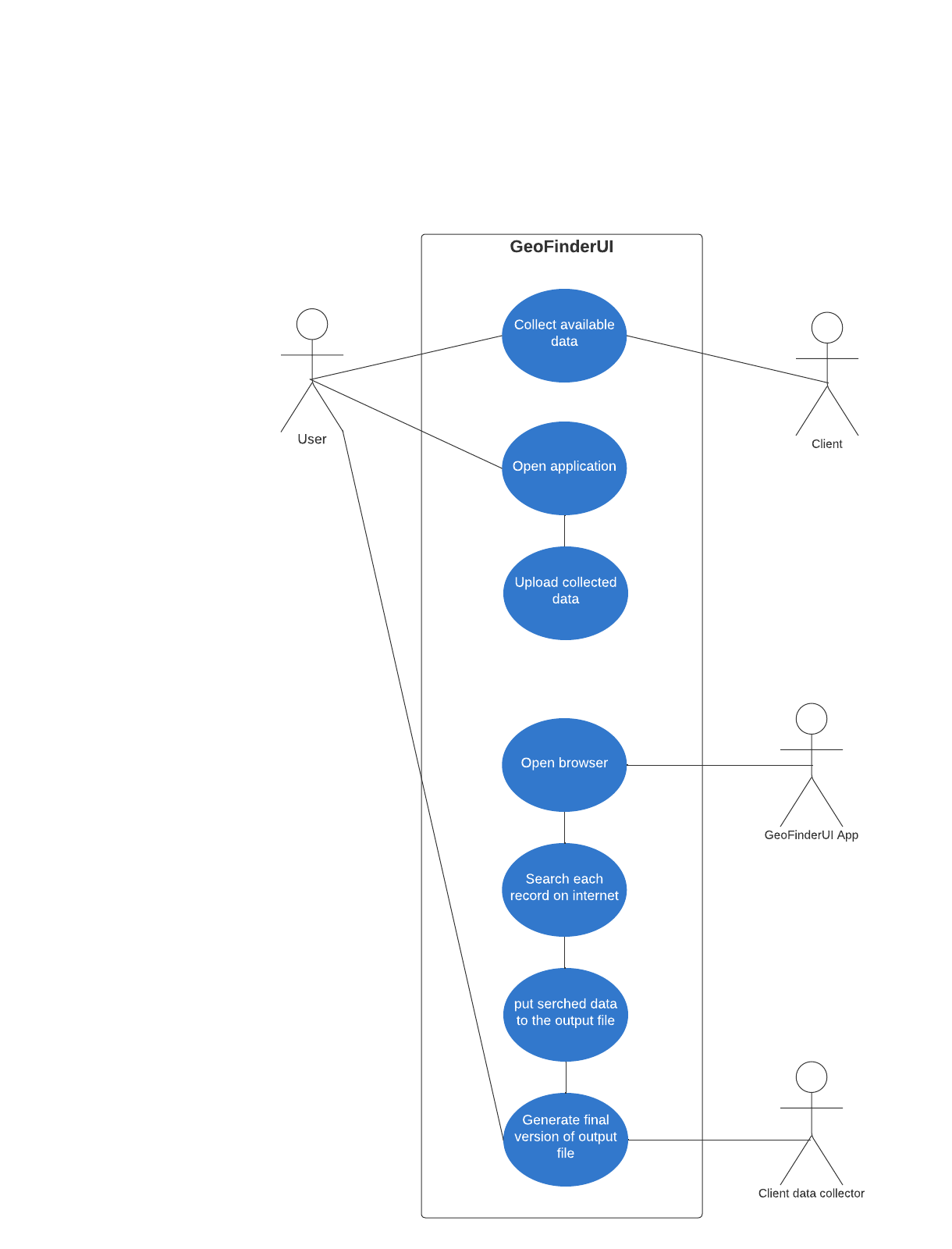
**Figure 11 data acquisition points by GeoFinderUI application**

After collecting such extensive information and documents that deepen the problems of the assumed task, the process of programming the application can begin.



**Figure 12 Manual geo finder use case diagram**

The process of searching for geolocation data carried out manually would be a time-consuming and exhausting process for the person responsible for collecting the data. All elements of the full process, by which we obtain geolocation data, would have to be performed by the user. Starting with the joint collection of input data with the customer. The next steps are up to the user only. These elements are finding a suitable location on the Internet on the basis of the input data provided, and then finding the missing information about the individual place whose data we entered for the search.



**Figure 13 GeoFinderUI use case diagram**

Unlike the manual process of obtaining data on the basis of selected input data about a specific location, the "GeoFinderUI" application independently and automatically searches, collects information and delivers it to a file for suggestive result data.

# Implementation.

## Front-end code

Front-end code is the part of the code responsible for all the elements visible on the page or modelling it.

### Home page

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>GeoFinderUI</title>  
 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>

<style>  
 html {  
 height: 100%;  
 }  
 body {  
 min-height: 100%;  
 display: flex;  
 flex-direction: column;  
 }  
 .content {  
 flex: 1;  
 }  
 footer {  
 margin-top: auto;  
 display: block;  
 bottom: 0;  
 width:100%;  
 height: auto;  
 }  
</style>

</head>

**Code 1 The <Head> section with the style part needed for positioning the content of the GeoFinderUI start page**

The entire user-visible part of the application was written in the "HTML" language. A large role in its appearance is played by styling packages provided by the portal "Materialize". With their help the site acquired a professional and user-friendly appearance. This would not be possible without the addition of 2 lines of code, which are references to the location of the saved styling packages "CSS" and "JS". The references were placed at the top of the source code of the website in the "<head> </head>" section. In this section you can also see the subsection "<style> </style>". The style section, prepared in this way, allows you to arrange the elements on the web page in a universal way with respect to the size of the window in which the page is displayed. When the size of the screen displaying the page is so small that the elements of the page overlap, a scroll bar is displayed on the right side of the window, making it possible to read any information on the page. The "<footer> </footer>" subsection has been given stylistic attributes, forcing this subsection to be displayed in the lowest possible position on the page, but not further away than required by the space provided for the proper display of all elements of the website. The same set of settings can be found on each of the existing subsections of the "GeoFinderUI" application.

<table class="table">  
 <thead>  
 <tr>  
 <th>Data File - template</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr>  
 <td>  
 <a href="/download\_template"><button class="waves-effect waves-light btn black-text">Download file here</button></a>  
 </td>  
 </tr>  
 </tbody>  
</table>

**Code 2 Part of the code that allows the user to download the official version of the data template acceptable by the application**

In order to provide the best services of the application, a link has been placed on the site that allows downloading the official template file in effect when the user submits data to the application. The link initiates the launch of a dedicated script for downloading files located on the site, written in the "Python" programming language.

<form enctype="multipart/form-data" action="/upload\_files" method="post">  
 <div>  
 <h4> <b>Upload data file: </b></h4>  
 <p><input type="file" name="excelfile" accept=".xlsx, .xls"/></p>  
 </div><br>  
 <div>  
 <input type="submit" id="proceed-xlsx-file" class="btn waves-effect waves-light" value="Proceed the file">  
 </div><br>  
 <div>  
 <h6>(Optional)</h6>  
 <h6>delay in sec:</h6>  
 <p><input class="col s1" type="number" step="0.5" min="1" name="delay"/></p>  
 </div>  
</form>

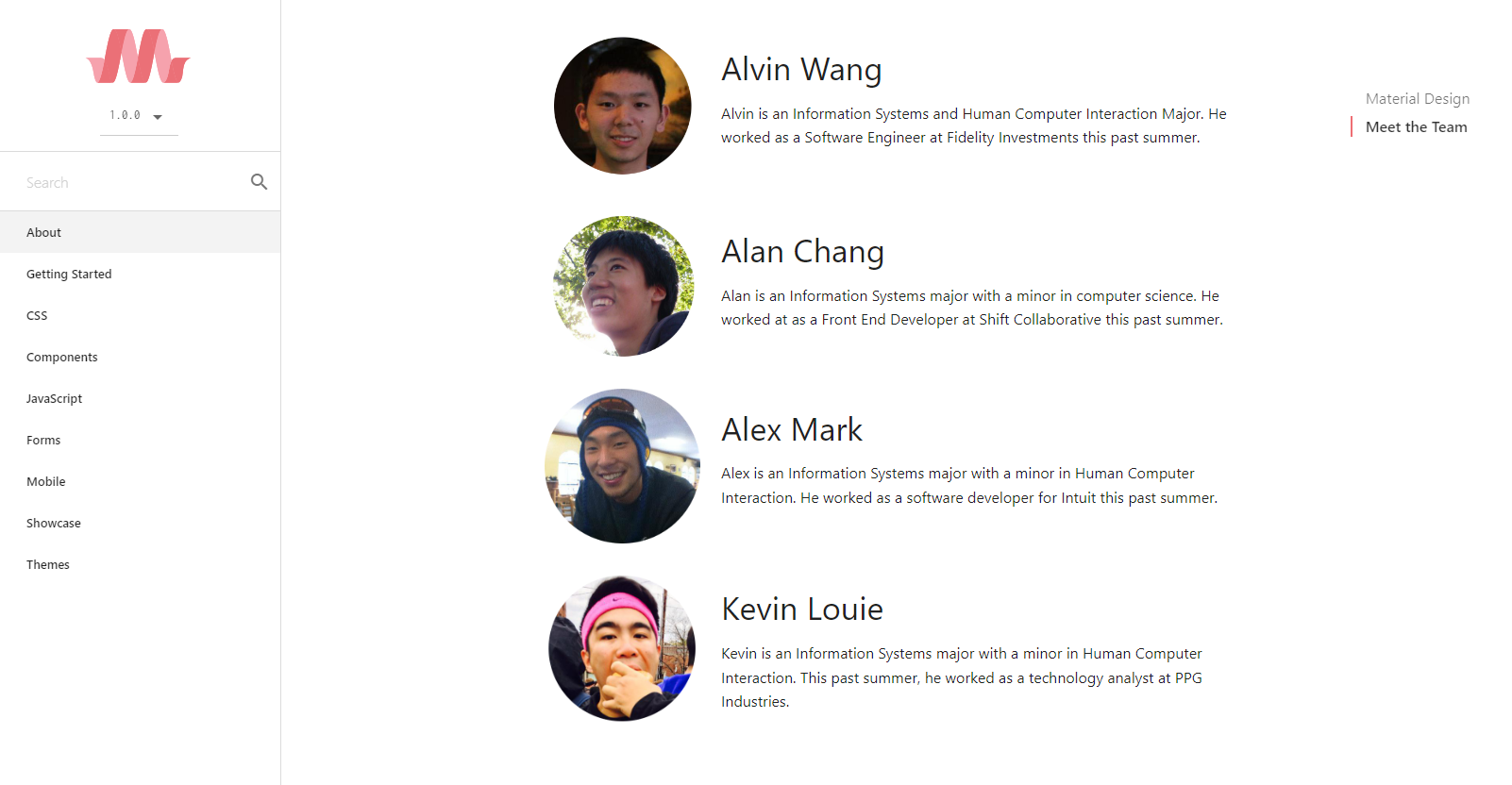
**Code 3 The form section needed for receiving and sending files and information to the back-end code**

In order to easily receive and process files or optionally provided information about the additional time needed to load the page, and then transfer them to the back-end code, the application uses the "<form> </form>" sections. With the help of this section, websites easily manage to provide the necessary information used in programming languages as variables. The "form" section contains 3 "<input>" fields, the first of which is used to upload the selected file by the user, the second approves the selections and starts the main functionality of the application, and the last is an unnecessary field informing the program about the delay in obtaining data in order to load the full pages.

<footer class="page-footer">  
 <div class="container">  
 <div class="row">  
 <div class="col s12 m12">  
 <h5 class="white-text">Additional information</h5>  
 <p class="grey-text text-lighten-4">Depend on your internet speed connection, &#013;declare the delay in sec for page to load, &#013;too low will occur in 'N/A' values. Recommended value: 1 =< 3</p>  
 </div>  
 <div class="col s12">  
 <h5 class="white-text">Visit my Websites</h5>  
 <ul>  
 <li><a class="grey-text text-lighten-3" href="https://www.facebook.com/m3mo.wujec/">Facebook</a></li>  
 <li><a class="grey-text text-lighten-3" href="https://www.instagram.com/m3mo\_w/">Instagram</a></li>  
 <li><a class="grey-text text-lighten-3" href="https://www.youtube.com/@m3mo\_w">Youtube</a></li>  
 </ul>  
 </div>  
 </div>  
 </div>  
 <div class="footer-copyright">  
 <div class="container">  
 © 2023 Copyright s20431/PJAIT  
 </div>  
 </div>  
 </footer>  
</body>  
</html>

**Code 4 The part of the code responsible for the footer visible on each page of the "GeoFinderUI" web application**

Each page displayed within the application "GeoFinderUI" has in it a part of the code responsible for generating a page footer. The footer located on the main page has additional information that cannot be found on any other page. This information is intended to explain to the user what the "delay in sec:" field is used for. Many sites available on the Internet have information about the author of the site or web application. The application "GeoFinderUI" also has information about the author in the form of external links to the author's community portals where information about the author's hobbies, interests and also information about the author's professional experience is published.



**Figure 14 Information about the authors of materializecss.com**

### Summary page

<div class="container">  
 <div class="row">  
 <div class="col s12 m12">  
 <div class="card blue-grey darken-1">  
 <div class="card-content white-text">  
 {% if message %}  
 <span class="card-title">{{ message }}</span>  
 {% endif %}  
 <p>As GeoFinderUI developers, we are extremely happy that you are using our tool.</p>  
 </div>  
 <div class="card-action">  
 {% if check %}  
 <a href="/download\_file"><button class="btn btn-default">Download file here</button></a>  
 {% endif %}  
 <a href="/home"><button class="btn btn-default">return to home page</button></a>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>

**Code 5 The part of the code that contains the information and its layout on the summary page**

The page devoted to the summary of the procedure performed by the "GeoFinderUI" application, has elements that can display other information depending on the result of the procedure performed. The values of the variables are passed to the display via a back-end script as parameters added to the "render\_template()" function from the toolkit that constitutes the "flask" library. The page is able to change its layout at the time of a positive result of data file generation or at the time of a negative result. When the result is negative, the information about will be displayed and the button allowing the user to download the file generated by a script written in the "Python" programming language is not available.

## Back-end code

### Initialization classes

import os  
import time  
  
from flask import Flask, Blueprint, render\_template, *request*, send\_file  
from selenium import webdriver  
from selenium.webdriver.common.keys import Keys  
from selenium.webdriver.common.by import By  
from selenium.common.exceptions import NoSuchElementException  
from selenium.common.exceptions import TimeoutException  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as *EC*import pandas as *pd*from openpyxl.workbook import Workbook  
from openpyxl import load\_workbook

**Code 6 Part of the code containing the import of the necessary tools used in the "GeoFinderUI" application**

The "home.py" file begins with a section of code devoted to importing libraries and the various tools needed to run the application.

@*home*.route('/')  
def index():  
 return render\_template('home.html')  
  
  
def download(*filepath*):  
 print(*filepath*)  
 return send\_file(*filepath*, as\_attachment=True)  
  
  
@*home*.route('/download\_template')  
def download\_template():  
 *path* = os.path.realpath('files/template\_files/Template.xlsx')  
 return download(*path*)  
  
  
@*home*.route('/download\_file')  
def download\_file():  
 *path* = os.path.realpath('files/download/S\_DataValues.xlsx')  
 return download(*path*)

**Code 7 Side methods in "home.py"**

The first methods found in the "home.py" file are side methods, used to initialize the start page after entering the corresponding "URL" address, and methods that allow downloading files such as "Template.xlsx" and a spreadsheet file holding the end results of the main application process "GeoFinderUI".

@*home*.route('/upload\_files', methods=["GET", "POST"])  
def upload():  
  
 *driver\_path* = os.path.realpath("webdrivers/chromedriver.exe")  
 *chrome\_options* = webdriver.ChromeOptions()  
 *chrome\_options*.add\_argument("headless")  
 *chrome\_options*.add\_argument("disable-gpu")  
  
 *path* = os.path.realpath('files/upload')  
 *path\_download* = os.path.realpath('files/download')

**Code 8 The beginning of the main method responsible for the functionality of the application**

The beginning of the main method responsible for the functionality of the application is the definition of variables necessary for the use of the script. There are functions that return the absolute path of a given location based on its fragment, and a variable that initializes the driver for the "Chrome" browser from "Google". Arguments have also been added to this variable to control the browser in the background so that the automated process of obtaining data from the browser does not appear on the user's screen.

if *request*.files:  
 if len(os.listdir(*path*)) != 0:  
 for *filename* in os.listdir(*path*):  
 *filepath* = os.path.join(*path*, *filename*)  
 os.remove(*filepath*)  
 if len(os.listdir(*path\_download*)) != 0:  
 for *filename* in os.listdir(*path\_download*):  
 *filepath* = os.path.join(*path\_download*, *filename*)  
 os.remove(*filepath*)  
  
 *upload\_file* = *request*.files["excelfile"]  
 *upload\_filename* = *upload\_file*.filename  
 *delay* = *request*.form.get("delay")  
  
 if *delay* == "" or *delay* == " ":  
 *delay* = 0.5  
 float(*delay*)

**Code 9 Part of the code responsible for preparing and cleaning the location for storing data files**

Before starting the process of searching the web pages in order to collect data forming the final result, the program analyses the target space for storing data files and when any file is found it is deleted, such procedure minimizes the risk of an error occurrence, also the location of such a procedure at the beginning of the code has a great impact on negating the potential occurrence of an error, because if the previous user had not brought the data acquisition process to an end, and the code fragment cleaning the location was located at the end of the procedure, the current user could have received results based on the data provided by its predecessor.

*data\_tab\_value* = pd.read\_excel(*data\_file\_path*, 'Data')  
*reader* = pd.read\_excel(*data\_file\_path*, 'Data', header=None, usecols="A:G")  
  
*wb* = load\_workbook(filename=*data\_file\_path*)  
*ws* = *wb*['Data']

*ws*['F1'] = "S\_Name"  
*ws*['G1'] = "S\_Address"  
*ws*['H1'] = "S\_City"  
*ws*['I1'] = "S\_Zip-code"  
  
*driver\_chrome* = webdriver.Chrome(executable\_path=*driver\_path*, options=*chrome\_options*)  
*driver\_chrome*.get("https://www.google.com/maps/")

**Code 10 Part of the code responsible for preparing the data file and launching the automated browser**

To ensure that the user is well informed about the values being entered into the spreadsheet, the program assigns header names that are a combination of the "S" corresponding to the word "Suggested" and the type of data being transmitted, such as address, city or zip-code.

for *record* in range(len(*data\_tab\_value*.index)):  
 *name\_value* = (*data\_tab\_value*.iat[*record*, 0])  
 *address\_value* = (*data\_tab\_value*.iat[*record*, 1])  
 *city\_value* = (*data\_tab\_value*.iat[*record*, 2])  
 *zip\_code\_value* = (*data\_tab\_value*.iat[*record*, 3])

**Code 11 A loop procedure that goes through all the records entered in the data file**

The whole application is based on the "for" loop, the program at this point collects data provided by the user, each iteration is another line in the spreadsheet.

if pd.isna(*name\_value*):  
 if *address\_value* and *city\_value*:  
 *s\_name* = name\_f\_address\_city(*driver\_chrome*, *search*, *delay*, *address\_value*, *city\_value*)  
 if pd.isna(*zip\_code\_value*):  
 if *s\_name* != "N/A":  
 *s\_zip\_code* = zipcode\_f\_name(*driver\_chrome*, *search*, *delay*, *s\_name*)  
 else:  
 *s\_zip\_code* = zipcode\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*)  
  
 elif *address\_value* and *zip\_code*:  
 *s\_city* = city\_f\_zipcode(*driver\_chrome*, *search*, *delay*, *zip\_code\_value*)  
 if *s\_city* != "N/A":  
 *s\_name* = name\_f\_address\_city(*driver\_chrome*, *search*, *delay*, *address\_value*, *s\_city*)  
  
if pd.isna(*address\_value*):  
 if *name\_value* and (*s\_address* == '' or *s\_address* == "N/A"):  
 *s\_address* = address\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*)  
 if pd.isna(*city\_value*) and *s\_address* != "N/A" and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_address(*driver\_chrome*, *search*, *delay*, *s\_address*)  
 elif pd.isna(*city\_value*) and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*)  
 if pd.isna(*zip\_code\_value*) and *s\_address* != "N/A" and (*s\_zip\_code* == '' or *s\_zip\_code* == "N/A"):  
 *s\_zip\_code* = zipcode\_f\_address(*driver\_chrome*, *search*, *delay*, *s\_address*)  
 elif pd.isna(*zip\_code\_value*) and (*s\_zip\_code* == '' or *s\_zip\_code* == "N/A"):  
 *s\_zip\_code* = zipcode\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*)  
  
if pd.isna(*city\_value*):  
  
 if *name\_value* and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*)  
 if pd.isna(*address\_value*) and (*s\_address* == '' or *s\_address* == "N/A"):  
 *s\_address* = address\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*)  
 if pd.isna(*zip\_code\_value*) and *s\_address* != "N/A" and (*s\_zip\_code* == '' or *s\_zip\_code* == "N/A"):  
 *s\_zip\_code* = zipcode\_f\_address(*driver\_chrome*, *search*, *delay*, *s\_address*)  
  
 elif *address\_value* and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*)  
 if pd.isna(*name\_value*) and *s\_city* != "N/A" and (*s\_name* == '' or *s\_name* == "N/A"):  
 *s\_name* = name\_f\_address\_city(*driver\_chrome*, *search*, *delay*, *address\_value*, *s\_city*)  
 elif *zip\_code\_value* and *s\_city* == "N/A" and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_zipcode(*driver\_chrome*, *search*, *delay*, *zip\_code\_value*)  
 if *s\_name* == '' or *s\_name* == "N/A":  
 *s\_name* = name\_f\_address\_city(*driver\_chrome*, *search*, *delay*, *address\_value*, *s\_city*)  
 if pd.isna(*zip\_code\_value*) and *s\_address* != "N/A" and (*s\_zip\_code* == '' or *s\_zip\_code* == "N/A"):  
 *s\_zip\_code* = zipcode\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*)  
  
 elif *zip\_code\_value* and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_zipcode(*driver\_chrome*, *search*, *delay*, *zip\_code\_value*)  
  
if pd.isna(*zip\_code\_value*):  
 if *name\_value* and (*s\_zip\_code* == '' or *s\_zip\_code* == "N/A"):  
 *s\_zip\_code* = zipcode\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*)  
 if pd.isna(*city\_value*) and *s\_zip\_code* != "N/A" and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_zipcode(*driver\_chrome*, *search*, *delay*, *s\_zip\_code*)  
 elif pd.isna(*city\_value*) and pd.isna(*address\_value*) is not True and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*)  
 if pd.isna(*address\_value*) and (*s\_address* == '' or *s\_address* == "N/A"):  
 *s\_address* = address\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*)  
  
 elif *address\_value* and (*s\_zip\_code* == '' or *s\_zip\_code* == "N/A"):  
 *s\_zip\_code* = zipcode\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*)  
 if pd.isna(*city\_value*) and *s\_zip\_code* != "N/A" and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_zipcode(*driver\_chrome*, *search*, *delay*, *s\_zip\_code*)  
 elif pd.isna(*city\_value*) and (*s\_city* == '' or *s\_city* == "N/A"):  
 *s\_city* = city\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*)  
 if pd.isna(*name\_value*) and *s\_city* != "N/A" and (*s\_name* == '' or *s\_name* == "N/A"):  
 *s\_name* = name\_f\_address\_city(*driver\_chrome*, *search*, *delay*, *address\_value*, *s\_city*)

**Code 12 Part of code with nested "if" statements that returns final data values**

The program has multiple nested conditional procedures in order to obtain the searched data in the most efficient way. Each data acquisition path is precisely prepared for all possible scenarios of data arrangement. The program is designed and written in such a way that there are 9 ways to get the next information about a given location from each of up to 4 pieces of information provided by the user.

data\_to\_file(*name\_value*, *s\_name*, *address\_value*, *s\_address*, *city\_value*, *s\_city*, *zip\_code\_value*, *s\_zip\_code*, *ws*, *record*)  
  
 *message* = "Success! - file is ready"  
 *check* = 'pass'  
  
 *driver\_chrome*.quit()  
 *wb*.save(*download\_file\_path*)  
 else:  
 *message* = "Please use the official version of the template file available on main page"  
 else:  
 *message* = "Please use the official version of the template file available on main page"  
 else:  
 *message* = "please reload page and make sure to choose the data file"  
 else:  
 *message* = "please reload page and make sure to choose the data file"  
else:  
 *message* = "upload filed!"  
return render\_template('summary.html', message=*message*, check=*check*)

**Code 13 Last part of main class code**

The last part of the main module of the application, enters the data collected by the program into a spreadsheet file and saves it under the name "S\_DataValues.xlsx". The script assigns to the variable "message" the final information about the executed process. The variable "check" is a tag used to generate a page containing a summary. If this variable has the value "pass", the user has access to a button that initiates the download of a file having the final data stored in a spreadsheet named "S\_DataValues.xlsx".

### Callable classes

def cookies(*driver\_chrome*):  
 *delay\_cookie* = 10  
 try:  
 myElem = WebDriverWait(*driver\_chrome*, *delay\_cookie*).until(EC.visibility\_of\_element\_located((By.XPATH, '/html/body/c-wiz/div/div/div/div[2]/div[1]/div[3]/div[1]/div[1]/form[2]/div/div/button')))  
 except TimeoutException:  
 print("Loading took too much time!")  
 try:  
 *driver\_chrome*.find\_element(by=By.XPATH, value="/html/body/div[2]/div[2]/div[3]/span/div/div/div/div[3]/button[2]/div").click()  
 except Exception:  
 pass  
 try:  
 *driver\_chrome*.find\_element(by=By.XPATH, value="/html/body/c-wiz/div/div/div/div[2]/div[1]/div[3]/div[1]/div[1]/form[2]/div/div/button").click()  
 except Exception:  
 pass

**Code 14 Class dedicated to accept cookies on website**

A class thanks to which a browser can approve a pop-up window that requires accepting the storage of cookies emitted by a website visited by the program. With the help of web scraping, the application locates the approval button and then it is remotely clicked.

def address\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*):  
 try:  
 *driver\_chrome*.find\_element(by=By.NAME, value='q').clear()  
 except Exception:  
 pass  
  
 *search*.send\_keys(*name\_value*)  
 *search*.send\_keys(Keys.RETURN)  
  
 page\_to\_load(*driver\_chrome*, *delay*)  
 time.sleep(1)  
 try:  
 *s\_address* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[7]/div[1]/button/div[1]/div[2]').text  
 except NoSuchElementException:  
 *s\_address*="N/A"  
  
 if *s\_address*=="N/A" or *s\_address*=="":  
 try:  
 *s\_address* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Copy address"]').text  
 except NoSuchElementException:  
 *s\_address*="N/A"  
  
 if *s\_address*=="N/A" or *s\_address*=="":  
 try:  
 *s\_address* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Kopiuj adres"]').text  
 except NoSuchElementException:  
 *s\_address*="N/A"

return *s\_address*

**Code 15 A class that returns address value from name value**

Acquiring an address from the place name specified by the user starts with entering the phrase into the search box and then validating it. When the correct value is not found, the program looks for the corresponding information in other locations on the page containing the search results. Then it returns the value assigned to the "s\_address" variable.

def city\_f\_zipcode(*driver\_chrome*, *search*, *delay*, *zip\_code\_value*):  
 try:  
 *driver\_chrome*.find\_element(by=By.NAME, value='q').clear()  
 except Exception:  
 pass  
  
 *search*.send\_keys(*zip\_code\_value*)  
 *search*.send\_keys(Keys.RETURN)  
  
 page\_to\_load(*driver\_chrome*, *delay*)  
 time.sleep(1)  
 try:  
 *s\_city* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[2]/div[1]/div[1]/div[1]/h1/span[1]').text  
 except NoSuchElementException:  
 *s\_city*="N/A"  
  
 try:  
 if *s\_city* != "N/A":  
 *temp\_city* = *s\_city  
 s\_city* = *temp\_city*.split()[1]  
 except Exception:  
 print("too fast")  
  
 if *s\_city*=="":  
 *s\_city*="N/A"  
 return *s\_city*

**Code 16 A class that returns city value from zip-code value**

Getting the city from the postal code is a very accurate way to determine the city in which the location is located. The value obtained from the search consists of the combination of the postal code and the city, for this purpose the program must split the retrieved text and assign to the variable "s\_city" the second part of the text, which has an index of 1.

def name\_f\_address\_city(*driver\_chrome*, *search*, *delay*, *address\_value*, *city\_value*):  
 try:  
 *driver\_chrome*.find\_element(by=By.NAME, value='q').clear()  
 except Exception:  
 pass  
  
 *search*.send\_keys(*address\_value* + " " + *city\_value*)  
 *search*.send\_keys(Keys.RETURN)  
  
 page\_to\_load(*driver\_chrome*, *delay*)  
 time.sleep(1)  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.CLASS\_NAME, value="bfdHYd Ppzolf")  
 *s\_name* = *s\_name*.get\_attribute("aria-label")  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[2]/div[1]/div[1]/div[1]/h1').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[15]/div/div[2]/div[2]/div[1]/div/div/div/div[1]/div/span').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[17]/div/div[2]/div[2]/div[1]/div/div/div/div[1]').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[17]/div/div[2]/div[2]/div[1]/div/div/div/div[1]').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[19]/div/div[2]/div[2]/div[1]/div/div/div/div[1]/div').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[15]/div/div[2]/div[2]/div[1]/div/div/div/div[1]/div').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[27]/div/div[2]/div[2]/div[1]/div/div/div/div[1]/div').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="N/A" or *s\_name*.upper()==*address\_value*.upper():  
 try:  
 *s\_name* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[18]/div/div[2]/div[2]/div[1]/div/div/div/div[1]/div').text  
 except NoSuchElementException:  
 *s\_name*="N/A"  
  
 if *s\_name*=="":  
 *s\_name*="N/A"  
 return *s\_name*

**Code 17 A class that returns name value from address value and city value**

The class that returns the name of the location is one of the most extensive classes used in the "GeoFinderUI" application. Its extensiveness results from several possible positions of the information sought on the page subjected to the technique of searching the source code, web scraping. The source code of the "Google Maps" pages does not have unified tags for the elements found on the search results page. This is problematic, so the program must refer to the location on the page using "X-Path", which is the path of sequential tags such as "<div>" or "<span>" counted from the beginning of the source code of the page.

def city\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*):  
 try:  
 *driver\_chrome*.find\_element(by=By.NAME, value='q').clear()  
 except Exception:  
 pass  
 *exception\_check* = 0  
  
 *search*.send\_keys(*name\_value*)  
 *search*.send\_keys(Keys.RETURN)  
  
 page\_to\_load(*driver\_chrome*, *delay*)  
 time.sleep(1)  
 try:  
 *s\_city* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[2]/div[1]/div[1]/h2[2]/span').text  
 *exception\_check* = 1  
 except NoSuchElementException:  
 *s\_city*="N/A"  
  
 if *s\_city*=="N/A" or *s\_city*=="":  
 try:  
 *s\_city* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Copy address"]').text  
 except NoSuchElementException:  
 *s\_city*="N/A"  
  
 if *s\_city*=="N/A" or *s\_city*=="":  
 try:  
 *s\_city* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Kopiuj adres"]').text  
 except NoSuchElementException:  
 *s\_city*="N/A"  
  
 try:  
 if *s\_city*!="N/A" or *s\_city*!="":  
 *temp\_city* = *s\_city* if *exception\_check* == 0:  
 *temp\_city* = *temp\_city*.split(sep=",")[1]  
 *s\_city* = *temp\_city*.split()[1]  
 except Exception:  
 *s\_city*="N/A"  
  
 if *s\_city*=="":  
 *s\_city*="N/A"  
 return *s\_city*

**Code 18 A class that returns city value from name value**

The class that assigns and returns the value of the "s\_city" variable obtained by entering the location name in the search field is mainly based on finding the full address of that location. In this case, finding this information on the page is easier, because next to each search result for a field having an address value, there is a shortcut for users to copy the full address to the clipboard in the device's memory. Due to the possible occurrence of the source code and the elements contained therein in Polish or English, the criteria for finding the appropriate item on the page are repeated, but having the word value "Copy address" in both languages. The next step is to split the received value of the full address and extract from it information about the city in which the location is located.

def zipcode\_f\_name(*driver\_chrome*, *search*, *delay*, *name\_value*):  
 try:  
 *driver\_chrome*.find\_element(by=By.NAME, value='q').clear()  
 except Exception:  
 pass  
 *exception\_check* = 0  
  
 *search*.send\_keys(*name\_value*)  
 *search*.send\_keys(Keys.RETURN)  
  
 page\_to\_load(*driver\_chrome*, *delay*)  
 time.sleep(1)  
 try:  
 *s\_zip\_code* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[2]/div[1]/div[1]/h2[2]/span').text  
 *exception\_check* = 1  
 except NoSuchElementException:  
 *s\_zip\_code*="N/A"  
  
 if *s\_zip\_code*=="N/A" or *s\_zip\_code*=="":  
 try:  
 *s\_zip\_code* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Copy address"]').text  
 except NoSuchElementException:  
 *s\_zip\_code*="N/A"  
  
 if *s\_zip\_code*=="N/A" or *s\_zip\_code*=="":  
 try:  
 *s\_zip\_code* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Kopiuj adres"]').text  
 except NoSuchElementException:  
 *s\_zip\_code*="N/A"

try:  
 if *s\_zip\_code*!="N/A" or *s\_zip\_code*!="":  
 *temp\_zip\_code* = *s\_zip\_code* if *exception\_check* == 0:  
 *temp\_zip\_code* = *temp\_zip\_code*.split(sep=",")[1]  
 *s\_zip\_code1* = *temp\_zip\_code*.split()[0]  
 *s\_zip\_code* = *s\_zip\_code1* except Exception:  
 *s\_zip\_code*="N/A"  
  
 if *s\_zip\_code*=="":  
 *s\_zip\_code*="N/A"  
 return *s\_zip\_code*

**Code 19 A class that returns zip-code value from name value**

Similar to the class that obtains the city in which a location is located from the name of that location, it uses the location's full address to then split it and assign under the "s\_zip\_code" variable a value corresponding to the location's zip code contained in the full address of the searched location.

def city\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*):  
 try:  
 *driver\_chrome*.find\_element(by=By.NAME, value='q').clear()  
 except Exception:  
 pass  
  
 *search*.send\_keys(*address\_value*)  
 *search*.send\_keys(Keys.RETURN)  
  
 page\_to\_load(*driver\_chrome*, *delay*)  
 time.sleep(1)  
 try:  
 *s\_city* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[2]/div[1]/div[1]/div/div[2]/div[2]/div[1]/div/div/div/div[4]/div').text  
 except NoSuchElementException:  
 *s\_city*="N/A"  
  
 if *s\_city*=="N/A" or *s\_city*=="" or *s\_city*.upper() == *address\_value*.upper():  
 try:  
 *s\_city* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Copy address"]').text  
 except NoSuchElementException:  
 *s\_city*="N/A"  
  
 if *s\_city*=="N/A" or *s\_city*=="" or *s\_city*.upper() == *address\_value*.upper():  
 try:  
 *s\_city* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Kopiuj adres"]').text  
 except NoSuchElementException:  
 *s\_city*="N/A"

try:  
 if *s\_city*!="N/A" or *s\_city*!="":  
 *temp\_city* = *s\_city  
 s\_city* = *temp\_city*.split(",")[1]  
 *temp\_city* = *s\_city  
 s\_city* = *temp\_city*.split()[1]  
 except Exception:  
 *s\_city*="N/A"  
  
 if *s\_city*=="":  
 *s\_city*="N/A"  
  
 return *s\_city*

**Code 20 A class that returns city value from address value**

The class which obtains the name of the city in which a given location is located on the basis of its address, uses the same way of obtaining information as in previous classes, by full address of the location, and then splitting the text value and extracting from it the searched value, which in this case is the name of the city in which the location is located.

def zipcode\_f\_address(*driver\_chrome*, *search*, *delay*, *address\_value*):  
 try:  
 *driver\_chrome*.find\_element(by=By.NAME, value='q').clear()  
 except Exception:  
 pass  
  
 *search*.send\_keys(*address\_value*)  
 *search*.send\_keys(Keys.RETURN)  
  
 page\_to\_load(*driver\_chrome*, *delay*)  
 time.sleep(1)  
 try:  
 *s\_zip\_code* = *driver\_chrome*.find\_element(by=By.XPATH, value='//\*[@id="QA0Szd"]/div/div/div[1]/div[2]/div/div[1]/div/div/div[2]/div[1]/div[1]/div/div[2]/div[2]/div[1]/div/div/div/div[4]/div').text  
 except NoSuchElementException:  
 *s\_zip\_code*="N/A"  
  
 if *s\_zip\_code*=="N/A" or *s\_zip\_code*=="" or *s\_zip\_code*.upper() == *address\_value*.upper():  
 try:  
 *s\_zip\_code* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Copy address"]').text  
 except NoSuchElementException:  
 *s\_zip\_code*="N/A"  
  
 if *s\_zip\_code*=="N/A" or *s\_zip\_code*=="" or *s\_zip\_code*.upper() == *address\_value*.upper():  
 try:  
 *s\_zip\_code* = *driver\_chrome*.find\_element(by=By.CSS\_SELECTOR, value='[data-tooltip="Kopiuj adres"]').text  
 except NoSuchElementException:  
 *s\_zip\_code*="N/A"  
  
 try:  
 if *s\_zip\_code*!="N/A" or *s\_zip\_code*!="" or *s\_zip\_code*.upper() != *address\_value*.upper():  
 *temp\_zip\_code* = *s\_zip\_code  
 s\_zip\_code* = *temp\_zip\_code*.split(",")[1]  
 *temp\_zip\_code* = *s\_zip\_code  
 s\_zip\_code* = *temp\_zip\_code*.split()[0]  
 except Exception:  
 *s\_zip\_code*="N/A"  
  
 if *s\_zip\_code* == "" or *s\_zip\_code*.upper() == *address\_value*.upper():  
 *s\_zip\_code*="N/A"  
  
 return *s\_zip\_code*

**Code 21 A class that returns zip-code value from address value**

To get the postal code of a given location using the given address of that location, the program uses the same way as the previous described class. The way is to get the full address, and then split it and extract a piece of text corresponding to the postal code of the location.

# Summary

## Application outcome

The application "GeoFinderUI" is a program that returns to the user suggestive data about a given location, which were obtained from the minimum input data provided by the user. All search results are subject to the risk of incorrect data, this is due to the program finding the first, most matching search result to the input data entered. The idea behind this application is to reduce as much as possible the time needed to obtain geolocation data required in a defined business environment, which requires obtaining specific output data. The application was successfully formulated and written, thus achieving a fully functional program. Thanks to this project, I have achieved a much broader perspective on IT topics, mostly related to programming. My knowledge has been enriched with a lot of new information, skills and experience.

## Acknowledgments

The most interesting and rewarding part of the application development process was the achievement of the set goals. Thanks are due to Maksymilian Kubica, M.Sc. for his professional supervision of the project and for providing many valuable tips during the process of writing the thesis. The knowledge and experience he imparted are invaluable, and his management of the project allowed it to develop properly. For providing a place to work, help during brainstorming, and support, thanks are due to all those close to me - parents and friends.

# Bibliography

* *Aplikacja internetowa*. (2022, December). From Wikipedia: https://pl.wikipedia.org/wiki/Aplikacja\_internetowa
* *Bot program*. (2022, May). From Wikipedia: https://pl.wikipedia.org/wiki/Bot\_(program)
* *Code Refactoring*. (2023, January). From Refactoring: https://refactoring.com/
* *Discover Intellij Idea*. (2023, January). From jetbrains: https://www.jetbrains.com/help/idea/discover-intellij-idea.html
* Flower, M., Beck, K., Opdyke, W., & Brant, J. (2002). Refactoring Helps You Find Bugs. In M. Flower, *Refactoring: Improving the design of Existing Code* (pp. 48-49). Addison-Wesley.
* *Framework*. (2022, October). From Wikipedia: https://pl.wikipedia.org/wiki/Framework
* *Google Maps*. (2022, May). From https://www.google.com/maps/about/#!/
* *HTML*. (2023, January). From https://html.spec.whatwg.org/
* *Markup language*. (2023, January). From Wikipedia: https://en.wikipedia.org/wiki/Markup\_language
* *Materialize*. (2023, January). From https://materializecss.com/
* *Microsoft Excel*. (2023, January). From Wikipedia: https://en.wikipedia.org/wiki/Microsoft\_Excel
* *Openpyxl*. (2022, May). From https://openpyxl.readthedocs.io/en/stable/
* *Pandas software*. (2023, January). From Wikipedia: https://en.wikipedia.org/wiki/Pandas\_(software)
* *Python*. (2023, January). From https://docs.python.org/pl/3/
* *Selenium*. (2022, December). From https://www.selenium.dev/
* *Uniform Resource Locator*. (2022, December). From Wikipedia: https://pl.wikipedia.org/wiki/Uniform\_Resource\_Locator
* *Web scraping*. (2023, January). From Wikipedia: https://en.wikipedia.org/wiki/Web\_scraping

# Attachment

## Table of Figures

[**Figure 1 Presentation markup example** 8](#_Toc127208177)

[**Figure 2 Procedural markup example** 9](#_Toc127208178)

[**Figure 3 Descriptive markup example** 10](#_Toc127208179)

[**Figure 4 File Template.xlsx** 13](#_Toc127208180)

[**Figure 5 Input file with test data** 13](#_Toc127208181)

[**Figure 6 Output file with test data** 14](#_Toc127208182)

[**Figure 7 Google Maps search result** 23](#_Toc127208183)

[**Figure 8 GeoFinderUI application home page with Materialized styling packages**.......................................................................................................................24](#_Toc127208184)

[**Figure 9 the original diagrammatic process of the operation of the GeofinderUI application** 26](#_Toc127208185)

[**Figure 10 sample visualization of the GeoFinderUI start page** 27](#_Toc127208186)

[**Figure 11 data acquisition points by GeoFinderUI application** 27](#_Toc127208187)

[**Figure 12 Manual geo finder use case diagram** 28](#_Toc127208188)

[**Figure 13 GeoFinderUI use case diagram** 29](#_Toc127208189)

[**Figure 14 Information about the authors of materializecss.com** 34](#_Toc127208190)

## Table of Pictures

[**Picture 1 Left side, Martin Flower Picture 2 Right side, William Opdyke** 7](#_Toc126793112)

[**Picture 3 Left side, IntelliJ dark theme Picture 4 Right side, IntelliJ default light theme** 13](#_Toc126793113)

[**Picture 5 Left side, Nyan Progress Bar Picture 6 Right side, Snyk add-in** 14](#_Toc126793114)

[**Picture 7 Left side, Excel Worksheet Picture 8 Right side, Visual Basic for Applications** 15](#_Toc126793115)

[**Picture 9 Left Side, CWI building Picture 10 Right side, Guido van Rossum** 16](#_Toc126793116)

[**Picture 11 Jason Huggins** 17](#_Toc126793117)

[**Picture 12 Wes McKinney** 18](#_Toc126793118)

[**Picture 13 Left side, Eric Gazoni Picture 14 Right side, Charlie Clark** 19](#_Toc126793119)

[**Picture 15 Left side, Timothy Berners-Lee Picture 16 Right side, Robert Cailliau**.........................................................................................................................20](#_Toc126793120)

## Table of Code

[**Code 1 The <Head> section with the style part needed for positioning the content of the GeoFinderUI start page** 26](#_Toc126793275)

[**Code 2 Part of the code that allows the user to download the official version of the data template acceptable by the application** 27](#_Toc126793276)

[**Code 3 The form section needed for receiving and sending files and information to the back-end code** 28](#_Toc126793277)

[**Code 4 The part of the code responsible for the footer visible on each page of the "GeoFinderUI" web application** 29](#_Toc126793278)

[**Code 5 The part of the code that contains the information and its layout on the summary page** 30](#_Toc126793279)

[**Code 6 Part of the code containing the import of the necessary tools used in the "GeoFinderUI" application** 31](#_Toc126793280)

[**Code 7 Side methods in "home.py"** 32](#_Toc126793281)

[**Code 8 The beginning of the main method responsible for the functionality of the application** 32](#_Toc126793282)

[**Code 9 Part of the code responsible for preparing and cleaning the location for storing data files** 33](#_Toc126793283)

[**Code 10 Part of the code responsible for preparing the data file and launching the automated browser** 34](#_Toc126793284)

[**Code 11 A loop procedure that goes through all the records entered in the data file** 34](#_Toc126793285)

[**Code 12 Part of code with nested "if" statements that returns final data values** 36](#_Toc126793286)

[**Code 13 Last part of main class code** 37](#_Toc126793287)

[**Code 14 Class dedicated to accept cookies on website** 38](#_Toc126793288)

[**Code 15 A class that returns address value from name value** 39](#_Toc126793289)

[**Code 16 A class that returns city value from zip-code value** 39](#_Toc126793290)

[**Code 17 A class that returns name value from address value and city value** 41](#_Toc126793291)

[**Code 18 A class that returns city value from name value** 43](#_Toc126793292)

[**Code 19 A class that returns zip-code value from name value** 44](#_Toc126793293)

[**Code 20 A class that returns city value from address value** 45](#_Toc126793294)

[**Code 21 A class that returns zip-code value from address value** 47](#_Toc126793295)

## What CD contains

* **\Szymon\_Wujec\_s20431\_DiplomaThesis\_EPG**
  + Electronic version of engineering thesis
* **\GeoFinderUI**
  + All files necessary for the functioning of the web application