PHP Web Scraper for Regular Expressions

Thursday 2nd January, 2020 - 10:46

Flavio De Jesus Matias
University of Luxembourg
Email: flavio.dejesus.001@student.uni.lu

Giacomo Di Tollo
University of Luxembourg
Email: giacomo.ditollo@unive.it

***Abstract*—This document presents the bachelor semester project of Flavio De Jesus Matias under the tutoring of Giacomo di Tollo. The project consists of a web-based scraper, which allows users to search a website for regular expressions. Using the web interface, the user can specify the criteria which are converted to a condition which can be evaluated using PHP. This condition is then matched on the given website and the user can export the number of occurrences as a CSV file.**

1. Introduction

Web scraping plays a big role in the digital business industry nowadays. Web scraping is the use of a crawler, which can be defined as an internet bot, to retrieve information from an external website. This bot consistently browses trough the internet and searches for new specific pieces of information based on a defined topic. In other words, data scraping is the process of extracting data from a different website which is later going to be analysed for commercial or personal use.

An interesting use of a web scraper is a price comparison website which scrapes multiple e-stores for a specific product, fetch the price of the respective article and rank them from lowest to highest. This is a good way to use web scraping for the average customer because they can simply choose to buy the product from the cheapest website saving tons of money [2].

Modern e-commerce's have adopted this kind of techniques to adapt to their respective competitors and can, as described above, monitor a competitor's price changes on a certain product and compare it to their price. This allows them to lower their price dramatically and achieve **two** different things: get new customers and drive the competition out of the market. This pricing strategy can also be considered as *'predatory pricing'*, which is an **illegal** act violating the antitrust law of multiple countries. The problem with this method is that it is very difficult to prove in court [3].

Machine learning is another field where web scraping is important. Since machine learning has a high demand for data, web scraping is used to gather the needed information across millions of websites. An example of machine learning is that many banks use complex algorithms to identify loan risk and approve or deny the loan to the customer based on the conclusion only.

There is a lot of discussion about whether web scraping is legal or not. The main crucial point is how the scraped data is used afterwards and because there are very different ways to use the data, it is impossible to say if it's illegal or not. Scraping itself is not illegal (except it is said so in the 'Terms of use' and Copyright details of the targeted website), but the malicious use of the data is.

2. Project description

This section introduces the domain and objectives of the project.

2.1. Domain

The domain of this project is to develop and use an algorithm to retrieve occurrences of regular expressions to assess whether a website is dealing with a given topic. For this purpose, the search algorithm and a web-based application were created. To create a website, there are are two different parts which contain code: client and server-side.

- Client-side: The client-side, also called the front-end, contains the Web interface. This portion of the website contains the code which is visible to the user. From here the user is able to send a request to the server-side part.

- Server-side: The server-side, also called the back-end, handles each user's request and processes a response. This portion of the website contains the code which is **not** visible to the user.

2.1.1. Web interface. In order to create a Web application, a Web interface is required. This interface belongs to the client-side and allows the interaction between the user and the software that is running on the server, i.e., it sends requests to the server-side part. It is also called the User-Interface (UI). To create this User-Interface, three different languages were used: HTML (a markup language), CSS (a style sheet language) and JavaScript (a scripting language).

2.1.2. HTML. HTML (HyperText Markup Language) is a markup language which is used to build the structure of a webpage. HTML code consists of multiple nestable tags. These tags normally come in pairs, i.e. there are at least **two** of the same tag because they must be opened (**e.g.** <tag>) and closed (**e.g.** </tag>). Every HTML tag has a different goal, such as defining divisions, implementing images, audio or videos, defining headers, etc.

2.1.3. CSS. CSS (Cascading Style Sheet) is a style sheet language which describes how the HTML tags are to be presented. Every HTML tag can be customized with CSS. Each CSS tag consists of **two** parts: the selector and the declaration block.

Example:

$$\underbrace{\text{h2}}_{\textbf{selector}} \quad \underbrace{\{\text{color: red; font-size: 20px}\}}_{\textbf{declaration block}}$$

The selector is the HTML tag to be customized. The declaration block is surrounded by braces and consists of multiple property-value pairs separated by semicolons.

```
h2 { color: red; font-size: 20px }
```

Figure 1: Example CSS code

2.1.4. JavaScript. Similar to the two other programming languages mentioned above, JavaScript is also a client-side language. This object-oriented programming language allows the creation of interactive effects and animations on a website. JavaScript, also called a scripting language, has the capacity to implement APIs (Application Programming Interfaces) which gives the developer a wide range of choices between ready-made code parts. There are **two** different sorts of APIs: browser APIs and third-party APIs.

2.1.5. PHP. PHP (Hypertext Preprocessor) is like JavaScript a scripting language, but the main difference between them is that JavaScript is a client-side language and PHP is a server-side language. PHP is used to create Web Applications. In order to run PHP code on a local or remote server, it must be installed, unlike other programming languages. Only a web browser is required to execute a PHP file.

2.2. Objectives

There are mainly **two** goals in this project. The first and main goal consists of developing and using an algorithm to retrieve occurrences of regular expressions to describe the content of a website. The second goal is to build a web-based scraping interface to allow the user to interact with the algorithm.

In other words, the goal of this project is to build a scraper and use it to be able to describe the content of websites based on individual conditions.

2.2.1. Algorithm to retrieve occurrences of regular expressions. With the help of PHP libraries, an algorithm to retrieve occurrences of regular expressions was created. This algorithm must be able to transform a website's source code into meaningful text, translate user-input conditions into machine code and execute it on the identified text fully respecting the number of levels indicated at the start. It then should calculate the number of matches and output the results as a CSV file.

2.2.2. Web-based scraping interface. To conclude, an interface was not dispensable and was created using the computer languages mentioned above. This interface allows the user to interact with the machine code which runs on the back-end. This way the user can input a specific website's URL and create his wanted condition for regular expressions that is later executed on the server.

3. Background

The scientific part of this project consists of understanding regular expressions to be able to create a specific algorithm to identify them on website's source code while the technical part focuses on the creation of a Web application and the research for PHP libraries, which will help during the development of the algorithm. Before starting the project, knowledge in these multiple programming languages was mandatory and the understanding of regular expressions too. A lot of research about web scraping and where it is used was also done in order to understand the project and know what exactly was needed to accomplish the goals set at the beginning.

3.1. Scientific

Regular Expressions

Regular Expressions, also called Regex, are patterns (or filters) that can manipulate text strings. Creating such a specific pattern allows the user to extract exact information from a string or text file by accepting a certain part of the string matching the filter and ignoring the rest. This pattern consists of multiple characters, metacharacters and operators [4].

3.2. Technical

3.2.1. PHP. The most challenging part of this project was the use of the scripting language PHP. This part of the project requires more advanced knowledge in this programming language in order to develop an algorithm that extracts website's source code. To facilitate the creation of the algorithm, PHP libraries were used. PHP libraries are collections of code which contain a variety of classes and functions whose goal is to solve common problems. The library used in this project is '*Html2Text*'. More details about it will come later.

3.2.2. HTML & CSS. The basics needed to create a web interface are HTML and CSS. This means that it was mandatory to have knowledge in both these languages in order to complete this part of the project.

3.2.3. JavaScript. To create multiple input boxes dynamically and send the information via AJAX to the server, JavaScript was needed. This allows the user to create an infinite long condition and send the information to the server without reloading the website.

4. Algorithm to test the occurrences of a topic on a website

This section contains the scientific deliverable of the project. The scientific question which is defined as whether we can use an algorithm to identify if a specific website is talking about a given topic or not will be treated and answered in this section.

4.1. Functional requirements

The functional requirements are the tasks that the program should accomplish on doing, i.e. the behaviour of the product.

4.1.1. Input & Output. In order to function correctly, the algorithm needs input given by the user. This input consists of the URL(s) and the condition. Without this input, the algorithm is unable to work since it does not have any data to analyze. To give the user the results back, the algorithm needs to output the occurrences back as a response to the interface. These results need also to be printed to the result CSV file.

4.1.2. Web Server Application. This algorithm was developed using the programming language PHP, which means that the best option was to install a local web server to run the complete website. The student chose the application called '*XAMPP (Apache)*' which facilitates the creation of the project environment since it is more of a beginner, simple-to-use program, where every needed component is installed in one package and automatically configured. This program is also delivered with PHP.

4.2. Non-functional requirements

The non-functional requirements are features that are not necessarily required but do contribute to a better algorithm.

4.2.1. User-friendly interface. To facilitate the interaction of the user with the algorithm, an interface was created. This interface is kept attractive and simple. In order to eliminate all uncertainties about the functionality of the website, it is constructed in a very minimal way.

4.2.2. Display results during execution. To better illustrate the scraping process, the interface will display the URL and the corresponding matches on the interface in real-time. This allows the user to directly see the results and to know how many URLs have already been scraped based on their input.

4.3. Design

The server-side code was designed in such a way that every file has its own purpose and functions. This prevents the code from being a complete mess and being badly organized. Using this system gives the code the needed structure and hinders the algorithm from being one file containing a big block of code. There are only **3** files of code from which 2 are classes.

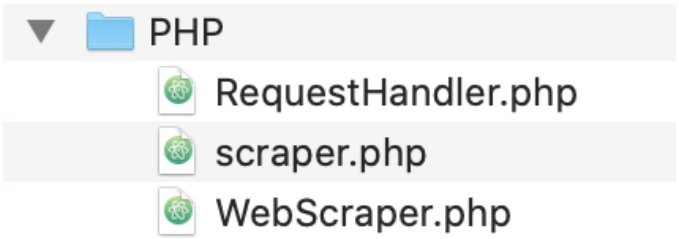

Figure 2: File structure of the server-side code showing the 3 files

4.3.1. WebScraper.php (Class). The WebScraper-Class is the most important file, it is the brain of the system. Every URL is an instance of this class in order to be scraped since this is the algorithm. The class will take the given URL and request the source code from it. The source code needs to be formatted in such a manner that the uninteresting parts are filtered out. This data is then manipulated using multiple functions to satisfy the specific needs of the scraper. First, all the URLs found in the source code are filtered out and considered 'sub-URLs'. The text is then split into multiple sentences while another function treats the condition building. This condition is then applied to each of these sentences and the number of occurrences of this regular expression is then stored for later output.

4.3.2. RequestHandler.php (Class). In order to handle the requests from the user, this class was created. It is a 'helper'-class for the 'WebScraper'-class. The first job of this class is to initialise all the required sessions on the very first scrape. The chosen scraping type is then treated in order to fill the sessions with all the desired URLs. The main goal of this class is to think in the user's place and to start the scraping process with the information provided which means to initialise the 'WebScraper'-class every time. It then builds the output to send back as a response to the interface which contains the current scraped data and most importantly the information to start the next request. In other words, this class knows which URL is being scraped and also which URL will be the next to be scraped, at all time.

4.3.3. Scraper.php. Similar to the 'RequestHandler'-class, this file was only created to facilitate the communication between the user and the algorithm, since it would be very difficult otherwise. This file is the only one the interface interacts with and is therefore responsible to accept the request and send the response back. This file just takes the data contained in the request and passes it to the 'RequestHandler'-class. It then calls all the needed functions from that class and finally returns the response back to the interface.

4.4. Production

To explain the production of each of these files each file and its functions will be explained in more detail, i.e. how they were produced, below.

4.4.1. WebScraper.php (Class). The WebScraper-Class contains the scraping algorithm itself. When initialising the class with an URL, a request is made to reach the URL. The request will then be categorised by the HTTP status code. This code indicates if the request was successful or not and if the request takes more than 15 seconds, it breaks off. The response containing the websites source code is then formatted with several methods. The main one is with the PHP library '*Html2Text*'. This library takes the source code of a website as an input and outputs the text from the website back without all the uninteresting code parts. The URLs contained in the source code are then written inside of brackets inline with the rest of the text. With the help of this data, there are multiple functions that will then continue the procedure.

1) **getHtmlAsArray**
   Using the processed text, this function then removes all the special characters that can affect the quality of the sentence recognition. All the URLs inside the brackets are removed as well. Then using a complex regex pattern, the sentences inside of the text block are identified as accurate as possible. These sentences are put inside of a list which is then returned by the function.

2) **createCondition**
   This function will take the condition passed by the interface and split it into multiple parts. The key words 'AND', 'ANDNOT', 'OR' and 'ORNOT' will then be replaced by the corresponding connectors in PHP. The condition is then put back together using these connectors and returned as a PHP-code line inside a string.

3) **calculateMatches**
   Using the two functions described above, this function loops through the whole list of sentences and apply the condition. Every time the condition matches in the given sentence, a variable is incremented. This variable is the number of matches obtained in the end.

4) **renderSubURLs**
   This function will identify all the URLs contained in the code which are inside of brackets. These URLs are then put together inside a list which is then returned by the function.

4.4.2. RequestHandler.php (Class). The RequestHandler-class is the class that handles the user's requests. This means that the class will prepare the next URL to be scraped, setup the levels, format the output that is going to be received by the interface and orders the scraping to start. To explain these procedures better, it is easier to explain the different functions and their behaviour. These are not all the functions, but the most important ones that do almost all the work.

1) **startApp**
   This function is only called at the very first request, which means it is called only once. The goal of this function is to initiate every session required (every URL and calculation is stored in sessions during the scraping process) and set up the app in order to run successfully. This means that it will recognize the chosen type by the user and either create a list with only one URL (type URL) or create a list with every URL included in the CSV file (type file). This list containing all the main URLs will then be used to continue the scraping process.

2) **selectURL**
   Every URL has a unique ID in the created list before. Using this ID and the current level, one can identify which URL is meant on the list. This function sets the current URL (the URL which is going to be scraped in the same request) to the one identified in the list.

3) **handleURL**
   This function uses the current URL (which was just set before by the 'selectURL'-function) and creates an instance of the WebScraper class. This represents the beginning of the scraping for this particular URL.

4) **handleLevels**
   According to the number of levels indicated by the user in the beginning, this function will take every URL contained in the list and get the respective sub-URLs. These URLs will then be added to another sub-list inside the main list denoted by '*lvl**n***', with *n* being the next level. To be more precise, if the number of levels indicated is 1, this function will not do anything. If the number is 2, this function will take every main URL and insert its respective sub-URLs inside the 'lvl2'-list. Otherwise, if the number is 3, it will also take all the URLs inside the 'lvl2'-list and insert it's sub-URLs inside another list called 'lvl3'.

5) **nextRequest**
   This function will take the current level and URL ID and check whether there is a next URL or not. If this

is the last URL for this level, the function then checks if there is another level. If there is no level left, this means that it was the very last URL and the 'continue' variable is set to false. If there is another URL left, this function will output the information (ID and level) about the next URL. This information will then be used during the next request in the 'selectURL'-function.

6) **createOutput**
This function will output the current URL and the matches count. It will also use the returned data from the 'nextRequest'-function to create a JSON-formatted output. This output will be the server response to the interface. The figure 4 shows an example of how such a response looks like.

4.4.3. Scraper.php. This file acts as the middleware between the user and the algorithm. The information found in the request is split apart and used to initialise the 'RequestHandler'-class. The job of this file is to call the necessary functions of the 'RequestHandler'-class in order to run the program. It detects if the current request is the very first one or not and if it is, the 'startApp'-function is executed. After each request, the JSON-formatted data from the 'createOutput'-function is used to give the response back to the interface. After the last request, however, the interface will send a final request to obtain the calculations, but only if the chosen type is URL.

4.5. Assessment

This section states whether the different requirements set have been successfully reached or not.

4.5.1. Input & Output. This requirement was that first of all the algorithm needs an input to work. This means that it cannot do his job without the needed information which is the URL and the condition. The goal of the algorithm is to output the occurrences of the given condition, which consists of regular expressions, on a specific website. As we can observe from the **3** computed statistics files, this is the case. Thus, we can answer the scientific question which is whether we can use an algorithm to identify if a specific website is talking about a given topic by saying that this algorithm successfully completes this task since the output of the algorithm is the number of occurrences of that specific regular expression on the website. This requirement was therefore achieved with success.

4.5.2. Web Server Application. This requirement was to install the needed software in order to run the project. The program used to run the web server was 'XAMPP'. This application was installed with success on multiple devices and the project runs as intended. Since the application handles the project correctly, this requirement was also achieved with success.

4.5.3. User-friendly interface. The goal of this requirement was to build a user-friendly interface. Using a simple layout of only two columns containing 3 boxes, the interface was kept as minimal as possible. This gets all the complications out of the way and leaves everything uncomplicated to use. The interface was beautifully designed and is attractive to the user's eye, which means that the website is user-friendly. This requirement was achieved with success.

4.5.4. Display results during execution. Lastly, the goal of this requirement was to always display the results of the current URL on the interface during the execution of the program. Since for every URL, a request is made and each request gets a response, the URL and matches count was included in the response part. This data is then used to be displayed on the interface as a 'URL & matches'-pair. More details about this requirement are in the technical deliverable section. It is explained how exactly this is made and how this provokes a loop to go through all the URLs. Since the results are displayed in real-time on the interface as intended, this requirement is as well achieved with success.

5. Web-based scraping interface

This section contains the technical deliverable of the project. Here will be explained what was required to build the interface and also in detail how it was designed and produced to satisfy the needs of the project.

5.1. Functional requirements

The functional requirements are the tasks that the program should accomplish on doing, i.e. the behaviour of the product.

5.1.1. Server communication. The interface's main job is to communicate with the server by sending him the information provided by the user. This means that the interface is the middleware between the user and the scraping algorithm and has, therefore, a very important duty. Thus, it is of great importance that the interface communicates with the server as intended because otherwise, the scraping will not produce the desired outputs.

5.1.2. Collecting the information & Server Request. To keep things simple, the user has to fill out the necessary information and click on a single button, 'Start'. This button has to do all the work for the user, which means that it should collect all the required information and store it. The condition is converted using a simple jQuery function to facilitate the user in the way he has to indicate the desired condition. All these pieces of information are then sent as a request to the server which does all the rest of the work by itself. This keeps things simple for the person who is using the scraping tool and prevents them from not knowing how to keep going.

5.2. Non-Functional requirements

The non-functional requirements are features that are not necessarily required but do contribute a lot to good user experience on the website.

5.2.1. Self-explanatory interface. One important part of the interface was the challenge to design a self-explanatory interface. This means that the user doesn't need any to very little explanations to understand the concept of the interface which was kept very uncomplicated and clean. Using multiple labels to describe the different input parts and big buttons the user is capable of understanding what information is expected in each section of the box and what function is hiding behind each button. To simplify the user interactions, the tools are all kept in one single place so prevent the user from having to search for their needs.

5.2.2. Responsiveness of the template. The responsiveness of the template contributes to great user experience. This means that no matter what device the user is using to access the website, the interface will work as expected. Large or small displays are not a problem.

5.3. Design

For the web scraper to be fully functional, an interface was needed. This interface allows the user to interact with the created algorithm in order to scrape the desired website.

The website is also kept very simple because it is only a one-page interface, which means there is only one page on which the user can access all the possible tools and functions. The background colour is set to #eff0f3, which is a light grey colour and in the top-centre part of the page the name of the application is written: 'PHP Web Scraper', which also represents the title of the project. The font used to write the title was '*Indie Flower*' and it has a font size of 70 pixels.

The rest of the page is divided into multiple boxes that have a very minimalistic design. Each individual box has its own title which is written using the font '*Montserrat*' and font size of 26 pixels. The title uses a larger font than the rest of the normal text and has a special border at the bottom which is dashed, this provokes the user to instantly identify each independent box and its content. There are only **3** boxes in the interface and each has a very unique purpose. Each of them is described in more detail below.

5.3.1. URL & Conditions. This box is the main part of the interface and is basically the only one the user interacts with. After loading the page there are only two simple buttons which allow the user to choose between two types: 'URL' and 'file'. The 'URL' type is basically the same as the type 'file', but the main difference is that by choosing the type 'file' the user has to upload a CSV file containing one or multiple URL's to be scraped. The type 'URL' only allows the user to scrape **one** main URL. On the contrary, the type 'file' lets the user upload a simple CSV formatted file which includes one or **multiple** URL's. The purpose of this type is clearly not to only have one URL in the file, but it works too.

After choosing the desired type, the rest of the box slides down in an animation revealing either the input box for the website's URL or the ability to upload the CSV file. The user then should be able to insert the website's URL or upload the CSV file. After completing these steps, a condition needs to be defined. This is done in a very simple way by letting the user input the word they want and choose the desired connector. The input does not necessarily need to be only a word, it can be a small sentence or a combination of words separated by a white-space as well. In order to create a more complex and long condition, the user can use the button 'Add condition' to add another part of the condition which has the same structure – 'input & connector'. After completing the condition, the individual has the possibility to change the number of levels to scrape which is always 1 by default. The number of levels indicates how deep the scraper goes, i.e. how often sub-URLs are considered. There are **three** types of levels from which the user can select from:

- **Level 1:** This level only considers the main URLs. This means either the only URL inputted at the beginning or only the URLs contained in the CSV file.

- **Level 2:** This level identifies all the URLs contained in the first URL an considers them as well.

- **Level 3:** This level does the same thing as level 2, but for every URL identified in each level 2 URL. It basically considers every URL identified in each URL found in the main URL.

When finished with all these settings, the user is now ready to run the program by hitting the 'Start'-button.

5.3.2. Results. The 'Results'-box represents the console of the program. After each URL is scraped, the result is added to the box including the number of matches for that specific URL. This allows the user to know which URL just got scraped and see the results in real-time. When the type of the current scrape is 'URL', then every URL that appears in the console is also added to the final results CSV file which can be exported. However, if the type is 'file', then every URL is still added to the console but only the main URL's are included in the results file. To the right of the title is the 'Export as CSV'-button which allows the user to export the results. The button is pre-defined to be disabled.

5.3.3. Calculations. The 'Calculations'-box outputs some statistics about the current scrape. These statistics vary from the type selected at the beginning. For the type 'URL', the box itself appears when the scraping is finished and shows of the calculations. The type 'file' puts the calculations into the exportable CSV file since there are multiple different URLs which means that the box itself is not displayed in the interface. There are 5 different calculations which are outputted:

- Hits on first page
- Total number of hits
- Total number of pages
- Hits on first page / Total number of pages
- Total number of hits / Total number of pages

5.4. Production

The production of the interface was the least challenging part of the project since the algorithm, i.e. the server-side code part was more difficult. The interface contains the client-side code part of the project which is the only code part that is visible by the user. To start the interface, multiple frameworks/libraries were needed. Here is a quick description of them:

1) **Bootstrap 4**
   Bootstrap 4 is an open-source framework which helps one to create fast, easy and responsive websites. It comes with a lot of pre-defined design templates such as buttons, forms, tables, etc. which can help to customize the interface in a modern way. Bootstrap also comes with an amazing grid system which was the best choice to use in order to also give the interface a pretty layout and a mobile-friendly look which is not required but always useful and a good thing to have.

2) **Font Awesome Icons**
   Font Awesome is a free toolkit which provides a set of icons used to illustrate an interface. This makes the website more user-friendly and more appealable with a clean and simple look.

3) **jQuery**
   jQuery is a free JavaScript library which allows one to write easier JavaScript code. It is mostly used to create animations on a website and also to add dynamic content to the interface (like in this project). There are tons of features with jQuery, but only the used ones in the project will be covered in detail.

The interface was started by setting a simple background colour in CSS, then the title was created inside a section. This section was centred and given some spacing on the top of 35px and the bottom of 16px. This spacing will push the title 35px from the top of the screen and the rest of the content which comes after the title, 16px to the bottom. With the help of Bootstrap, the layout was structured and defined: 2 columns with the same size. The container in which all the grid system is done automatically centres itself leaving some space on the left and right side. Inside of each column, the content is pushed 15px from the left and the right. This means that between the content from both columns we are left with 30px in space. This is enough to give it a nice look and to keep the content well organized in each section of the page. Using this grid system, there is an automatic detection of the device's display width and when the minimal length is reached, each column is resized to take up 100% of the width of the container. This means that we are left with 1 single column of 100% width instead of 2 columns with 50% width respectively. This is the magic behind a mobile-friendly web interface. :)

After setting up the grid system, the boxes had to be created. The width of a box is 100% relative to the parent (in this case the column which usually takes 50% of the container). The height is relative to the content. The background colour was set to white and the border was rounded to give it a more clean look. The title was created by setting the font size to a bit bigger than the content and by giving a bottom border which is dashed. After each box, there is 10px of space which will force the next box to start 10px under the previous box. Then each box was created individually.

5.4.1. URL & Conditions. As mentioned before, this is the main box of the interface and the only one with which the user can interact. The two starting buttons, as well as all the other components such as buttons, textbox, input box, etc. present in the interface, are designed using Bootstrap.

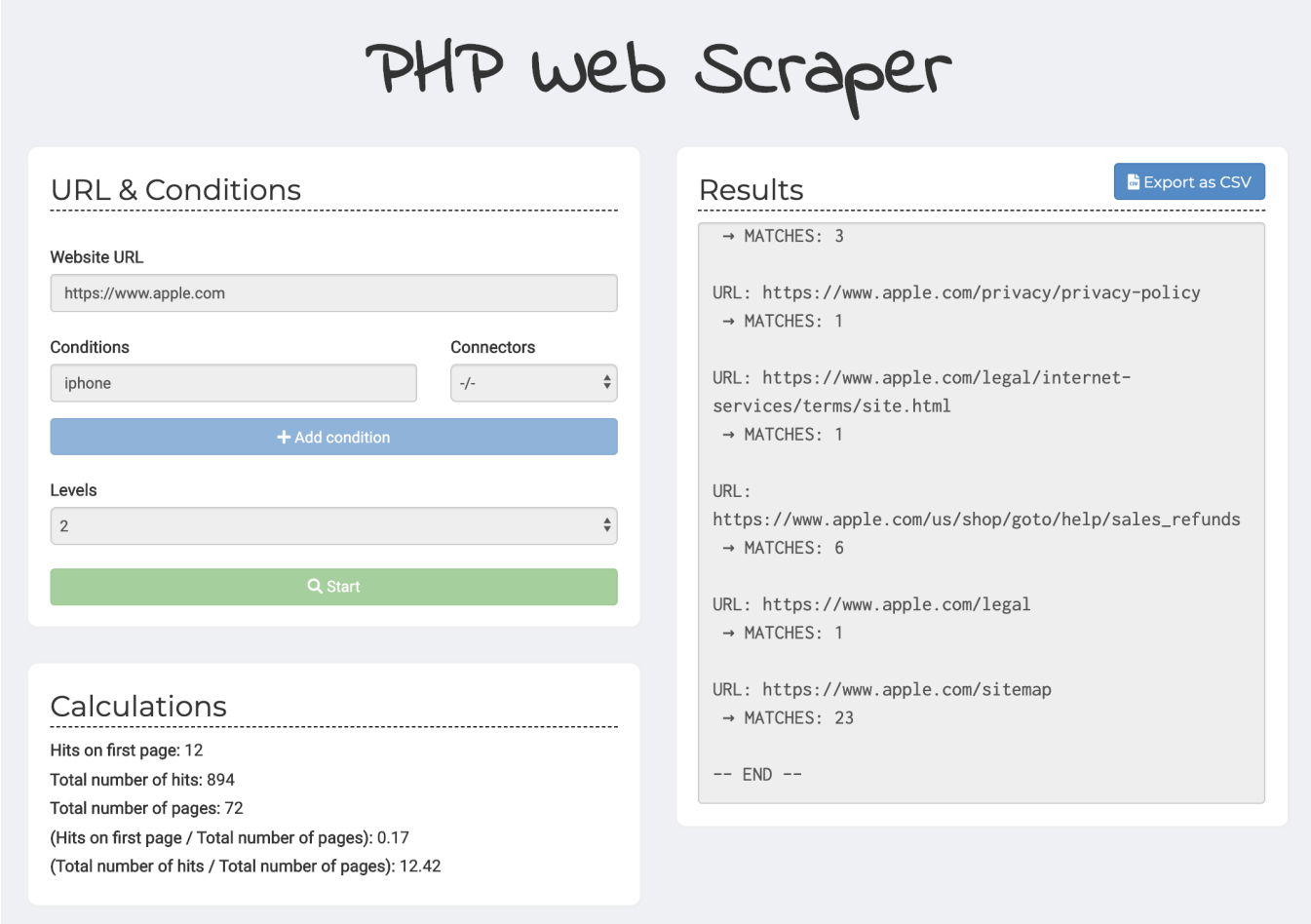

Figure 3: Screenshot of the interface after a scraping example

After clicking on one of these buttons, a jQuery function is executed which remembers the type the user chose, hides both of the buttons and slides down the rest of the box where all configurations are made. For the interface, the only difference between the two types is that there is either a simple input box or a file upload box which only accepts CSV formats. After that, there is the condition section. As already explained, this section consists of pairs with the same structure as – 'input & connector'. The input consists of a simple input box and the connector is done using the HTML 'select' tag which allows one to create dropdowns with multiple options. Below is the button 'Add Condition' which contains one of the three font awesome icons used. Clicking on this button, jQuery executes another function which will add another 'input & connector' pair dynamically below the last one. Afterwards is the levels selection. This is done with a simple dropdown where the user may select the level they want.

Lastly comes the most interesting part of the interface, the 'Start'-button. This is itself a simple button with an icon, but what happens behind it is what makes it interesting. By clicking on this button, a jQuery function is executed and all the input boxes and buttons are disabled. The goal of this function is to collect all the user input and store it. Using another function, the condition is formatted into a single line to later facilitate the understanding of the condition from the server. With this new condition and all the other input, this function sends a request to the server-side code (which is the algorithm) and gets a response. The response has multiple parts:

- (string) The current URL
- (int) The matches count for this URL
- (bool) Whether to continue or not
- (array) Information about the next URL (id & level)

Depending on the bool parameter, the function continues or stops. If the function continues, then it sets the next URL id and level to the ones obtained in the response and launches the same function (itself) again. Using the URL id and level, the algorithm can then identify the exact URL (which normally is the next in the list). This creates an infinite loop as long as the bool parameter is true.

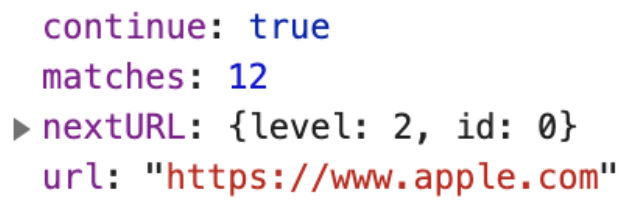

Figure 4: Example showing the multiple parts of the response from the server

5.4.2. Results. After each response from the server, the URL and matches are written in the 'console'. Therefore another jQuery function is needed which appends the text passed in by a parameter to the box. The 'console' itself is just a normal textbox created using HTML. This textbox was then disabled and the resize ability was set to 'none' (prevents the user from resizing the textbox manually) using CSS. The height and width were also pre-defined to prevent the box from resizing itself because of the dynamically added content to it. When the parameter is false, the function outputs '– End –' to the 'console' and stops. The 'Export as CSV'-button is then enabled and by clicking on

5.4.3. Calculations. The 'Calculations'-box only contains 5 rows of simple text. The numbers are pre-defined to be 0 and the box itself is hidden. If the type is 'URL' and the function stops, it sends a final request to the server. The response to this request contains all the calculations needed to display in the box. Finally, it replaces the 0's with the obtained values and shows the box to the user using jQuery.

5.5. Assessment

This section states whether the different requirements set have been successfully reached or not.

5.5.1. Server communication. The requirement was that the communication between the interface and the server works as intended. Using a jQuery Ajax call, the interface can send a collection of information as a request to the server and wait for a response. The algorithm will then handle the provided information and send a response back. This response is then yet again handled by the interface. This cycle was created successfully to work as long as not all the URLs are scraped.

5.5.2. Collecting the information & Server Request. The requirement was that the interface has sufficient resources to disposal in order to collect the information provided by the user. This is done using a JavaScript function which will look threw all the input boxes the user can manipulate. The collected information is then stored in a form and send to the server. This requirement was achieved with success.

5.5.3. Self-explanatory interface. To achieve this requirement, the interface needed to be self-explanatory. This means that the interface needed to be well structured, clean, simple and of course also attractive to the user's eye. As already explained in the scientific deliverable, this was successfully accomplished. Thus the requirement was achieved.

5.5.4. Responsiveness of the template. This requirement is based on the fact that the user can access the website using no matter what device (in terms of screen dimensions) and the interface will adapt itself to the screen size and that it stays beautiful and organized too. Using Bootstrap 4, this was not hard to accomplish. This was more of a bonus for the user than a challenge to complete around the interface. The requirement was therefore also achieved with success.

6. Conclusion

This paper presents a bachelor semester project whose goal is to use an algorithm to get the occurrences of a certain topic on a website with the help of a web interface. These kind of applications are faster, more accurate and more reliable when running in the console, but for the purpose of the project, an interface was created to visualize the process. However, the script was modified at the end to compute the multiple statistics files faster by running in the console.

There are multiple things that could still be done to improve the project in the future such as upload and run the project on a remote server. Since this is a pretty simple application, this wouldn't be very hard to achieve. After all, there are a lot of free web hosters which could have been used to upload the project, but this was not a major requirement.

Additionally, a user authentication system could be added using a database to allow the user to save his scraping progress.

Finally, using a machine learning procedure the algorithm could slowly improve its sentence recognition mechanism with the objective of augmenting the scraping quality of the final product. Although this is doable, it would be too advanced to achieve right now.

Acknowledgment

The authors would like to thank the BiCS management and education team for the amazing work done.

References

[1] Nicolas Guelfi: BiCS Bachelor Semester Project Report Template. https://github.com/nicolasguelfi/lu.uni.course.bics.global University of Luxembourg, BiCS - Bachelor in Computer Science (2017).

[2] Hiren Patel: How Web Scraping is Transforming the World with its Applications (article) https://towardsdatascience.com/https-medium-com-hiren787-patel-web-scraping-applications-a6f370d316f4

[3] What is web scraping? (Small article) https://www.imperva.com/learn/application-security/web-scraping-attack/

[4] Jan Goyvaerts: Regular Expressions, The Complete Guide. https://www.princeton.edu/ mlovett/reference/Regular-Expressions.pdf

7. Appendix

Material used to create the web-interface in this project:

1) Bootstrap 4:
   https://getbootstrap.com/

2) Icons:
   https://fontawesome.com/

3) Google Fonts:
   https://fonts.google.com/

4) jQuery (JavaScript library):
   https://jquery.com/