Dmitriy Haralson

Automating Website Crawling Using Web Scraping Techniques Provided by PHP

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Thesis

May 7th, 2016

Metropolia

Author Title Number of Pages Date	Dmitriy Haralson Automating website crawling using web scraping techniques provided by PHP 32 pages May 7th, 2016
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructors	Janne Rantala, Founder/Partner Jarkko Vuori, Principal Lecturer

The purpose of this study was to build a module for a web development application called Pro5. The module was designed to go to a potential customer's website and gather important information related to the page, such as page count and word count.

The project was realized in PHP using the Yii2 framework for any possible database connections. The use of web scraping techniques was also be required, and the built in DOM structure parsing tools were used to achieve this. The relational database management system that was used is MySQL. Due to the nature of this project, the style of the programming was very object oriented.

The results of the project were favorable as a module ready for integration into the Pro5 system was produced. The module also fairly accurately gathers data about the websites that it visits and scrapes.

Keywords	PHP,Web crawling, web scraping, Yii2

Tekijä Otsikko Sivumäärä Aika	Dmitriy Haralson Verkkosivujen indeksoinnin automatisointi käyttäen PHP:n verkkokaavinta tekniikoita 32 sivua 7.5.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Perustaja Janne Rantala Yliopettaja Jarkko Vuori

Insinöörityön tarkoituksena oli rakentaa moduuli nettikehitysalustalle nimeltä Pro5. Moduulin toimintaan kuuluu mahdollisen asiakkaan nettisivulla käynti ja tärkeiden tietojen keruu verkkosivulta, kuten sivujen ja sanojen määrä.

Moduulia toteutetaan PHP:lla käyttäen PHP:n päälle rakennettua runkokirjastoa nimeltä Yii2, josta hyödynnetään tietokantaominaisuudet. Verkkokaavintaosuuden mahdollistaa PHP:n sisään rakennettu DOM-rakenteen jäsennin. Tietokantatekniikkana toimii MySQL. Projektin luonteen takia ohjelmointityyli oli hyvin oliomainen.

Insinöörityössä toteutettiin integraatiovalmis moduuli Pro5-kehihitysalsutaan. Toteutettu moduuli kaapii melko tarkasti nettisivuja, joilla se käy.

Avainsanat	PHP, netti indeksointi, verkkoaavinta, Yii2

Contents

Acronyms

AJAX	Asynchronous JavaScript and XML. Used to make dynamic web pages.
API	Application Programming Interface
CGI	Common Gateway Interface
CLI	Command line interface
CRUD	Create, read, update and delete
HTML	Hyper Text Markup Language, the base language of any web application on the internet.
JSON	JavaScript Object Notation
MVC	Model-view-controller pattern
MySQL	The RDBMS that will be used in conjunction with Yii2 to store and manage data.
ORM	Object-relational mapping
PHP	Personal Home Page Hypertext Preprocessor
Pro5	The application into which the module that this document describes will go into.
RDBMS	Relational database management system
Seq 5 Oy	A web development company.
Yii2	A PHP framework that uses the MVC pattern as well as has ORM capabilities to get data from the database.

1 Introduction

As part of Seq 5 Oy's internationalization plan, the company asked me to a create web crawler for their new publishing environment Pro5. The goal of this web crawling tool is to make the life of a sales representative easier by speeding up the screening process of leads. Instead of making the sales representative make an educated guess of the price, the web crawler should, based on the data gathered, give an accurate price to the sales representative.

The main goal of the web crawler is to analyze a web page, and, as a consequence of this analysis, give that web page a price tag. Using this tool, a salesperson who does not know much about how a web page is made can easily and reliably, using this tool, tell the price of a web page without having the knowledge of the actual process that goes into the development of a/the page. This means that the web scraping done by the web crawler is really general, and indeed the goal was to make it work for 90% of the cases.

This thesis describes the planning process taken to make the web crawler. Furthermore, it describes the steps taken to achieve the best possible results given by the crawler through thorough discussion about the step taken to implement the web crawler. The thesis also brings up the performance of the web crawler. This includes how it fared in the wild, and possible performance bottlenecks on different internet connections.

2 PHP

2.1 A brief history

PHP was first created by Rasmus Lerdorf in 1993. He had to write a lot of CGI applications in the C language. This led him to take all of his code, which was already compiled into a library, and add a parser to it. This parser could read HTML files, and any possible custom tags inside the files. The parser could also call the C language functions, which then returned pure HTML, which a browser could read, and display to the end user. [1.]

The initial release of PHP (then called Personal Home Page Tools) was meant to only to track visits to Lerdorf's résumé that was on the internet. This original implementation did not have a native way to access databases. Eventually Lerdorf managed to add such a feature, and in 1995 he released the source code to the public for open source bug fixing, letting the public to add new features. [2.]

In 1997, Zeev Suraski and Andi Gutmans from Tel Aviv, Israel, talked to Lerdorf about rewriting the PHP core. The new version finally succeeded PHP 2 in the June of 1998. The new revision was so different from the original parser that it became its own language and looked like the modern day version of PHP, and was named PHP 3. At this point, PHP stood for the modern day recursive acronym, abandoning any reference to the limitations a tool designed for personal use might have. This version of PHP also added object-oriented programming features. [2.]

The biggest difference between PHP 3 and 4 is that PHP 4 introduced the Zend Engine. This engine was meant to improve performance of complex web applications. This meant that web applications using database connections, and over all, all the features that PHP 3 added, would be more efficient, and thus render the experience of a PHP-programmed web application more responsive. The initial release of PHP 4 was in 2000. [2.]

PHP 5, released in the July of 2004, introduced Zend Engine 2. This version improved significantly the object-oriented programming present in PHP prior to this point. At this point exceptions were also added to PHP. While initially the library of extensions that was available to PHP 5 did not have a large amount of usage of this feature, it slowly

grew as the uptake of PHP 5 increased. This meant that it now looked quite like C++, Java, and a host of other object-oriented programming languages. PHP 5 also saw improvements to other, already present, features, such as an improved XML parsing and creating tools. [2;3.]

Due to the fact that PHP was first compiled by a C programmer that made use of Lerdorf's C routines, it is no surprise that the code syntax is very C/Java-like. However, someone not familiar with Perl might overlook the fact that PHP also takes inspirations from there, namely the syntax of the variables. [2;4;5.]

On December 3rd, 2015, PHP 7 was marked for general availability. This new release brought a lot of needed syntactic sugar to PHP's table. On top of that, the release added options for the developer to make their code strongly typed. While this does not take the loosely typed mentality of PHP, it lets the developer decide whether they prefer the loosely typed style, of a strongly typed one. [6;7.]

2.2 Advantages of PHP

One of them main draws to PHP over other server side languages is the fact that it is free, but the language being free does not stop bigger companies from using it either, as PHP has powerful frameworks built upon it, such as Magento, which is a storefront framework used by big and small online stores alike. The companies that use Magento, and thus use PHP, include Samsung and Ford. The list of the frameworks does not stop there. There are many other frameworks, like Joomla. Joomla is a content management system. There are also other frameworks and systems written in PHP. This proves that PHP is a versatile system that can do a lot of things if proper programming logic is applied to it. [8;9.]

Having said that, the community of PHP is very large and active, and anyone can contribute to the source code of the language. This means that any question one might have either has been answered, or can be answered in a matter of minutes in a coherent manner. This also means that the documentation of PHP is extensive, and the documentation pages usually have examples given by the community, which also sometimes sparks discussion on the documentation pages. [8.]

The freedom that PHP offers also means that developers are not tied to a certain OS or a certain IDE. Seq5 Oy uses CentOS, a Red Hat redistribution, while I use a Windows environment for development. Code that was written in either of these development environments are compatible with one another, as evident by the difference in the OS technologies. Since PHP is a scripting language, one can also write code in a basic text editor, such as Windows' own Notepad editor. As with other proprietary server side languages, the developer is tied to a certain IDE [8.].

Another big draw of PHP is that there are prepacked tools that allow a developer to begin developing immediately. Installing these tools, of course is not necessary, as installing and configuring the PHP installation is a sounder option. However, if one need to demo something on a machine that does not a have a preconfigured installation of PHP, installing these tools makes for a quick demo environment, or starting development in a very timely manner. [10.]

2.3 Drawbacks

One of the biggest criticisms that PHP is receiving is the fact that the language, while being very C-like in the syntax of its functions, still manages to be very inconsistent in its naming conventions. However, Rasmus Lerdorf himself defends most of these naming conventions with the argument that a person skilled with MySQL and Linux systems will have an easy time using PHP. Furthermore, most of the major programs are written in C, and as such, the underlying API is C based. He does, however, also mention that the functions that did not have an API to reproduce are somewhat unconventional in the world of PHP. [10.]

Another significant drawback of PHP is that it is open source. This means that all of the code is available for all to see. This means that people with malicious intents can go into the code and can try to find exploitable errors. However, this can also be a good thing, as the same people can be searching for the errors and can patch them as soon as the errors are found. [11.]

Since PHP is a scripting language, there is no building mechanism, and thus no native way to hide the source code. This means that anyone who has access to the code can come and take the code and use it in their own project. Generally speaking, if the de-

velopment is for personal purpose and the code is kept private at all times, this should not be a problem, as in the end the developer is the only one who has access to the source code. However, in the case of making an application that will be deployed onto a customer's server, code that needs to be kept private will still be visible to anyone who has access to the server. Thankfully there are code obfuscation tools for PHP, so this drawback can be negated through external tools. [12.]

Generally speaking, PHP is not suitable for making desktop applications. Overall, any other language can handle this part of an application better than PHP. However, this this module will not be part of a desktop application, so this drawback is not necessarily one that needs to be taken into consideration. Besides there are multiple ways to make a PHP application look and feel like a desktop application. The drawback of this in and of itself is that one must bundle a PHP server with the application, or make it a pseudo desktop application and just edit a browser to look, feel, and act like a desktop application, when in reality it still gets all of its content from a server. [13.]

2.4 Reasons for choosing PHP

PHP was the language of choice because the already existing base for Pro5 was already written in PHP. The language was chosen to program the web crawling application described in this document. As things stand, this would mean an easier integration of the module into the already existing system, as opposed to wrapping the module's API, which was programmed in another language, with PHP. Also the project manager is proficient in PHP, so, should problematic situations arise, asking PHP related questions was easier and faster, as opposed to asking just general programming questions.

Another reason is that Seq 5 Oy employs really skilled frontend developers who boast a wide knowledge in HTML and CSS, and who are able to apply their extensive knowledge to responsive design using these technologies. Since PHP is a backend language, there must be a way to create user interfaces, and that way is generally HTML and CSS. That means that this language fills the need of the web crawling project itself, as well as the needs of other members on the team. This means that rapid development of the application can be ensured, both in the backend and the frontend.

PHP was also chosen because the mentioned system is built upon the Yii2 framework, which is an MVC ORM framework built upon PHP. Yii2 offers out of the box mapping of database tables to models, database migrations and the ability to make console applications. The framework also offers the ability to easily modularize the code through the framework's close link to a dependency manager called Composer.

The Yii2 framework also offers the ability to generate code for quick prototyping and the creation of views that can look like the final product. This also mean that a developer can create their own custom view templated for code generation. Not only does this speed up the development process, but also allows excellent customization options for their own code. This also mean that a developer can minimize the amount of repetitive tasks just by creating their own code generation template. This mainly applies to the views of the application, but if one programs the code generation properly, similar tasks in the future are rendered accessible through one press of a button.

The view generation ties into the ability to customize how a migration is generated, as the developer may also create a migration template for future use. The bottom line is that any aspect of the development process can be automated, which in turn saves development time in the long run.

3 Specifications

3.1 Basic functions of the web crawler

The main function that will become the back bone of the web crawler, is the ability to make a traversable site map automatically. This will be done through searching for, storing, and comparing the links that are present on a web site. The comparison is done on previously stored links to avoid duplicate entries. This process is the basic operation toward building a site map.

After the described workflow to construct a site map is implemented, each page will be further processed, measuring the complexity of each document on a website. The complexity is defined by the total amount of elements in a document, such as images and paragraphs. Generally speaking, anything that requires time to build and search for will be tied to some sort of a price tag, which will be shown to the end user, which in this case will be a sales representative in charge of the lead, and thus responsible for the initial contacts with the customer.

This is required for that one click evaluation of a website. This way a sales representative can have a little to no knowledge of both technical implementation and what the page itself looks like, and still give a price to a page presented to the sales representative. This will make getting more contacts out in a day easier and will possibly increase the revenue stream of a company.

Possible problems

One of the biggest problems that the web crawler will face are dynamic links. Dynamic links are ones that ae found on sites such as Facebook. In Facebook’s case, all the different pages are behind a dynamic link. This means, generally, that there is a template onto which the data is rendered. While useful for web application development, this can become a nightmare in web scraping, as the amount of data to be scraped just balloons.

This is why it is a good idea to be mindful of such a caveat when designing web scrapers, especially one such as this, as it is quite possible that a simple bug that was caused by poor cross referencing, might cause an infinite loop that fills the space of a

server, thus making any further development being done on the server at the same time grind to a halt.

A related problem to basic dynamic links, are AJAX enabled pages. This is where the infinite iteration can become a significant problem; however, once again, the proper planning and the right approach is instrumental to avoiding such a problem.

3.2 Advanced specifications

After the document that was downloaded from the customer's server is processed, the document will have to be stored. The save format will have to be so that this application is easy to expand upon, as the owner of Seq 5 Oy plans on expanding upon the base application that will be built here. One of the easiest things to do is just to make a text dump into a text file, but as one of the goals for the web crawling application after it is done is further development, a simple text dump is not enough. As the nature of this project is web scraping, while the web page is being parsed, a table structure must be built so that a site map will be built, which will serve as a stepping stone for further development of the application.

Another thing that must be taken into account is that a web scraping request can take multiple minutes to complete, depending on the complexity of a site. This means that this request will lock a session for the duration of the operation. This means that, while this will be a web application, another process must be opened for a seamless end user experience. Thankfully PHP offers console execution possibilities. Because of this Yii2 has a built-in application component. To avoid the mentioned deadlock, a console instance will have to be invoked through the initial response. For the duration of the scraping each website will have to have a status. The frontend will then have to query the server to see whether the status has changed to done, and then notify the end user of the completion of the operation. All of this can be done through database manipulation.

4 Planning

4.1 Basic structure

The basic structure will revolve around an MVC pattern. The main reason behind this decision is based to the fact that the Yii2 framework, which will be used in the web crawling project to access the database in this project, is a framework that utilizes this pattern. Most of the operations will be done on the model level, while the application entry points shall be through a controller, as the MVC pattern dictates. The views are quite an important part of this application, but as the data display function shall be plugged into another web application, using the widget function provided by Yii2 is the best way to go forward. A widget is a way of simplifying data representation by creating an API with settings for displaying data. This method, if planned out properly, provides a very flexible code reuse solution through reflection, as well as provides possibilities for displaying data in multiple ways through the use of multiple view files.

Below are the rules regarding how a web crawling application should behave when getting data from a website given by Greg Reda. These rules were, however laid out for a Python project in the field.

> Rules:
>
> You should check a site's terms and conditions before you scrape them. It's their data and they likely have some rules to govern it.
>
> Be nice - A computer will send web requests much quicker than a user can. Make sure you space out your requests a bit so that you don't hammer the site's server.
>
> Scrapers break - Sites change their layout all the time. If that happens, be prepared to rewrite your code.
>
> Web pages are inconsistent - There's sometimes some manual cleanup that has to happen even after you've gotten your data.
>
> [14.]

While these rules were laid out for a Python project, I believe the rules given here are universal to any and all web scrapers, including the one discussed here. The first two are probably the most relevant to the case of this web scraping tool. In the case of us-

ing the content on the website, Seq 5 Oy in the past has manually copy and pasted content from a customers' old website for demonstration purposes without a problem. The reasoning behind this is, in the end, that the display wrapper of the data is changed.

The second rule, on the other hand, is the one that this tool will have to adhere to. If it does not, there is a risk of creating a DDoS attack flag in the servers of the website, and possibly take the website down, an undesirable result when dealing with web scrapers. One possible solution is making the application wait for one to three seconds after every operation before sending a new request.

Instead of the web scraping capabilities of the web crawler being site specific, the capabilities will have to be general, meaning that the web scraper will have to be able to get data from almost any site. The initial goal is not to extract any actual data from the website accurately, but to count the amount of blocks of text, images, and links. Because of this, not all tags must be gone through. The most important ones are the <*img*> and the <*a*> tags. To calculate the amount of text will be significantly more difficult. It is not impossible, but to assume that all possible text will be in a <*p*> tag is not the correct course of action, as <*span*>, <*div*> as well as table and list structures are all possible locations of text. To use a regular expression in this case would just prolong the already possibly sluggish. The best option would be to strip all HTML tags and parse the document as pure text, as in count the total amount of words on one page. While this is not the best option, this will be able to give an estimate of the amount of work needed to do content creation for a given page. [15.]

Each page must be recursively and hierarchically gone thorough. This means that the process will be very time complex, as well as a very great possibility of memory complexity. Since the web scraper will have to go through every element, and HTML is generally structured as a tree the big O notation for the traversal of the body will be *O(n),* where *n* is the total amount of tags in the <*body*> tag. If the total amount of tags in the body is 1000, the script will have to iterate over itself 1000 times to get to the end of its lifecycle. It should be remembered that this is just one page of the website, so if there are 15 pages with 1000 tags, there will be 15,000 iterations.

Because of this, the planning stage of this project is of the most importance. Based on the scope of the project, the following class diagram has been made for the database tables.

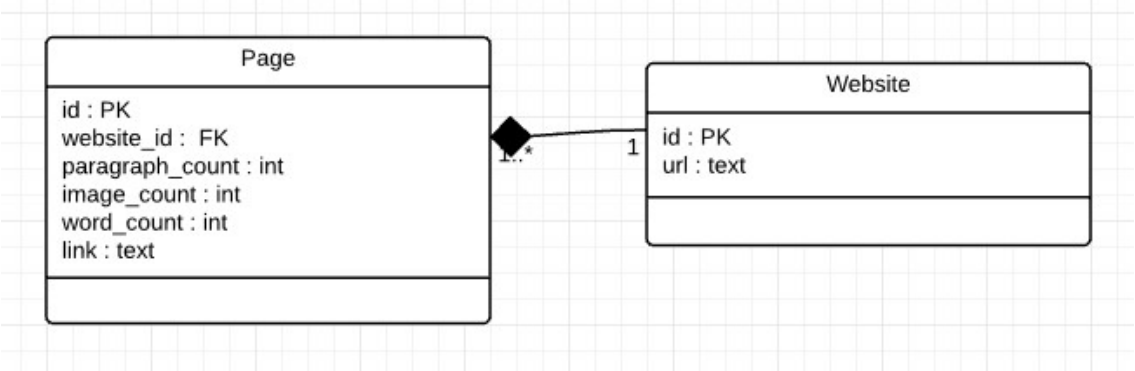

Figure 1: The class diagram for the database of the web scraping system.

Figure 1 denotes that there is at least one page for every website to be scraped. This relation is very logical, because, at its core, a website is a collection of pages, with at least one page, be it one landing page or a plethora of pages under one domain. However, there must be one page. This page will determine how the web scraping will proceed, as a search for links is conducted, as well as counting the amount of data the page has. Should a page have no internal links, we can safely say that there was one page, and end the scraping attempt.

The database maybe a simple many-to-one relationship, but the amount of rows in the page table will not be trivial as each website can have hundreds of pages. As such, when this tool is deployed, the amount of free space a server has to be monitored, and expanded accordingly should it run low on space.

4.2 Tools

4.2.1 Composer

Composer is a dependency manager for multiple platforms. It is primarily used through the command line interface for installing packages to a project, and should, if the packages being installed are configured correctly, install all of the required dependencies, removing any need to worry, whether all of the required packages for the library or

package to work are installed or not as this process is automated. This tool, therefore, allows for rapid development, and takes away the time wasted to set up a project. The Yii2 framework, and all of its components are heavily dependent on the Composer tool. Almost in all cases, the preferred way to install a package is Composer. Very little, if any, documentation exists on the subject of installing a Composer package by hand. Most of the questions regarding installation by hand are met with sarcastic comments, like “Use the composer install command” and “Use UNIX CLI”. This shows how easy the use of Composer really is. [16.]

4.2.2 Gii

Gii is a web based code generation tool, which can be installed through Composer. However, it comes packaged by default with the Yii2 application templates. Gii also come with its own API for a developer to use. This means that if the basic generation templates are inadequate, one can also extend and make their own generators. That being said, the web interface is enough for basic application building, as it sets up the basic structure of an application through a web interface.

It should also be noted that one can create view templates that the code generator will follow. This however is not required, and will only be used if the desired views differ greatly from the basic Yii2 templates. The project will use the framework’s base code, as it suits the development purposes.

4.2.3 Git

Git is an open source version control system. Git will be used to keep track of changes, as well as the version the source code. The Microsoft Windows installation package comes with two ways to control repositories: Git Bash and Git GUI. [17.]

Git Bash is a command line interface which supports most UNIX commands as well as Windows native commands, and comes configured with the Git command right at the developer’s fingertips. Using Git bash, however, is unnecessary, as the Git command can be used from the Windows CMD window, should the environment variables be properly configured. [18.]

Git GUI however is for the people who want to get into committing and pushing fast, and without prior knowledge of Git itself. Git GUI has all the Git commands in dropdown menus. It also shows which files have been selected for the commit as well as buttons corresponding to the written commands required to commit and push to a remote server. I used the Git GUI and found it to be very clunky for some tasks compared to the CLI counterpart, such as pulling and fast forwarding, while other features, such as committing and pushing, were very streamlined. That is why I suggest using both to complement both options' strengths and weaknesses.

The web interface that was used was used was GitLab Community Edition, which was hosted on a private server. This was the main way the project was kept secret, while still giving access to all the features that GitHub and GitLab has to offer.

4.3 Project workflow for the Yii2 framework

4.3.1 Setting up a project

Setting up a Yii2 for development purpose, of course, requires a PHP server. The choice of server for development does not matter, as long as the server runs a version of PHP 5.4 and newer, as Yii2 was built on top of this version of PHP. [19.]

Most of the plugins for Yii2, including the framework itself, can be easily installed through Composer. The developer must make sure that the path variables of the environment are correctly configured for the dependency manager to work correctly. However, if the installation happened through the executable file (at least on Windows systems), the environment paths will be set correctly, and Composer can be called from the console by name.

To install Yii2, it is necessary to enter the directory where the development of the application will happen with a command line interface, and enter the required command.

```
$ composer create-project yiisoft/yii2-app-basic web-crawler
2.x.x
```

Listing 1: Installing a basic application template through Composer

In listing 1, web-crawler is the name, or the directory, of the project, and 2.x.x is the version of Yii2 to be included in the project. The project can be created anywhere on the development computer, but the PHP server must have read permissions to the files, and the “web” directory that came with the installation of Yii2 must be placed in the public directory for that server. For demonstration purposes, the C:\ path of a Windows system will be used for the project location.

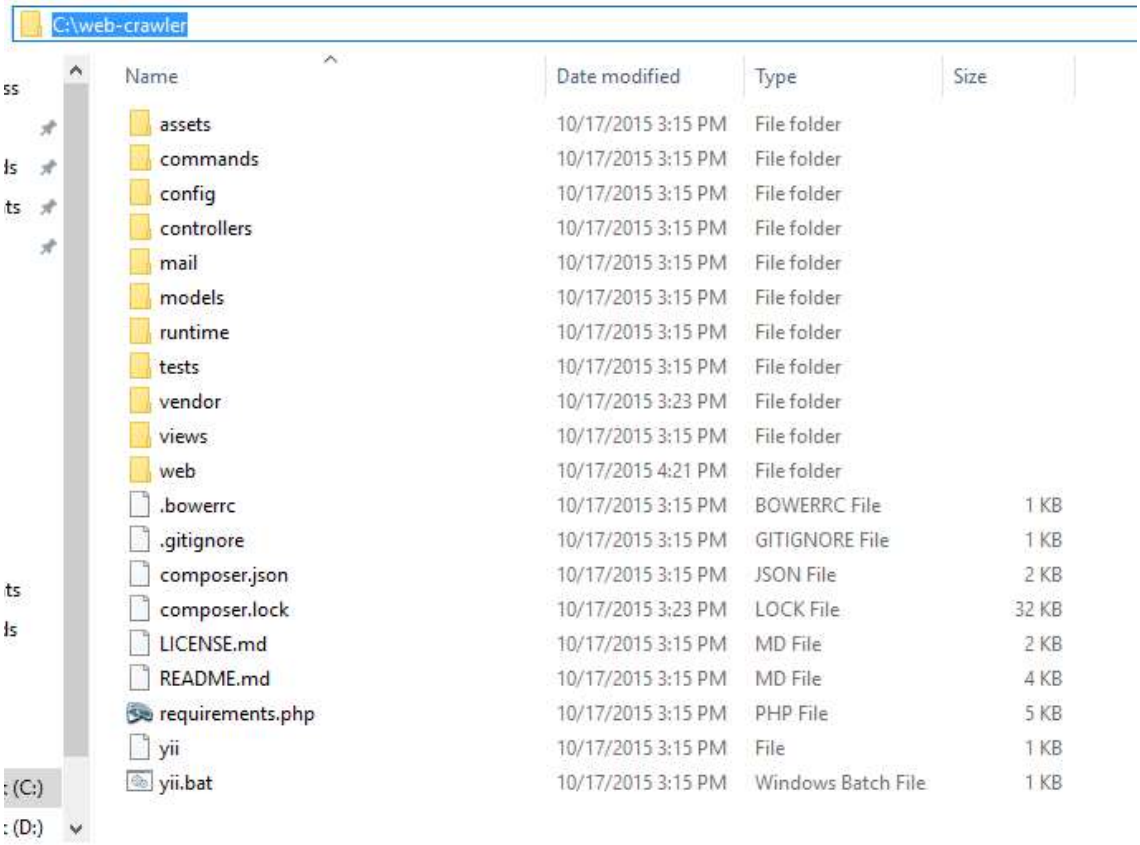

Figure 2: A fresh installation of Yii2 nn Windows Explorer.

Figure 2 shows what the project folder should look like on a successful installation of a basic template of the application. As one can see, there are three folders that are important at the start: models, views and controllers.

To make a server run the application itself, the previously mentioned “web” directory, or its contents, must first be moved to a server’s public_html or htdocs folder. For demonstration purposes, the default installation of XAMPP for Windows will be used. If the contents of the folder were moved to the public folder, entering the domain in the ad-

dress bar should be enough to enter the application. However, if the web folder was moved, it would be accessible through domain/web route. One can also rename the folder for different projects and separate applications. This demonstration will rename the web folder to “crawler” to differentiate this folder from the project folder in the C:\ directory. The final step is configuring the index.php that is inside the now renamed “web” directory.

```php
<?php

// comment out the following two lines when deployed to
production
defined('YII_DEBUG') or define('YII_DEBUG', true);
defined('YII_ENV') or define('YII_ENV', 'dev');

require(__DIR__ . '/../../../web-crawler/vendor/
autoload.php');
require(__DIR__ . '/../../../web-crawler/
vendor/yiisoft/yii2/Yii.php');

$config = require(__DIR__ . '/../../../web-crawler/
config/web.php');

(new yii\web\Application($config))->run();
```

Listing 2: A properly configured index.php file.

Notice from listing 2 that the debug mode of the application is enabled, then we are requiring the files central to the framework, which are now located in C:\web-crawler, and finally the application is started.

The file is configured to the demo case that has been outlined in the installation guide. Setting the project itself in this way is in no way a necessity and the default setup is also a viable way of setting up the project. However, this is the more secure option, as no one will be able to access any file publicly via a http request.

4.3.2 Migration

In theory, one should not need to use SQL or phpMyAdmin when creating a database. The reason behind this is the presence of a migration feature in Yii2. Migrations in Yii2 are executed, and can be created, from the CLI interface.

```
$ ./yii migrate/create create_tbl_page --fields=
website_id:integer:notNull:fk,paragraph_count:
integer:notNull,image_count:integer:notNull,word_count:
integer:notNull,link:text:notNull

$ ./yii  migrate/create create_tbl_website
--fields=url:text:notNull
```

Listing 3: CLI interface migration creation

The commands in Listing 3 will create the tables, and will help move databases from server to server, especially during development, when developers need fast access to new versions of a database. The CLI however currently has no option to create foreign keys, so they have to be added in the generated PHP file in the migrations folder of the application root. In this case, the generated PHP code will have to be edited.

```
$this->addForeignKey('website_id', 'tbl_page', 'website_id',
'tbl_website', 'id', 'CASCADE');
```

Listing 4: Adding a foreign key constraint with the CASCADE delete option

The code in listing 4 will add a foreign key with the CASCADE option, once the migration is run. The cascade option tells MySQL to delete all of the records related to the website table through website_id, should a parent record be deleted, thus preserving the integrity of the data in the database. The final step of the migration process is to commit it to the database.

```
$ ./yii  migrate
```

Listing 5: Invoking the created migrations

Listing 5 sets the created migration in motion, creating the tables and the required constraints. When the code will be versioned through Git, these migrations will be available to all developers, needing to only call the code snippet introduced here.

Should, however, the website table not have the URL column inserted during the creation of the tables, the missing column will have to be added. Because the create commands have already been executed, creating a migration where we create a new table would return an error, because the table with that name already exists. The correct solution for this situation would be to use the migration method addColumn.

```
$ ./yii migrate/create add_url_to_tbl_website
```

```
--fields=url:text
```

Listing 6: CLI way of creating a migration that adds a column

Executing the command in listing 6 will now add the missing column to the table which will create a line of code that uses the aforementioned addColumn method, negating the need to go into the phpMyAdmin interface, or executing an SQL query. If Git is being used, this will also be added to the repository. A collaborator will simply need to execute the same CLI command demonstrated in listing 5 to get the missing column imported into their database.

Once the migration has been executed, the database will have the tables we created using the CLI interface. Upon the execution of the migration, a migration database table is created. This tracks the executed migration files. Knowing this, we can create migrations for adding and removing columns as needed. This greatly increases the collaboration capacity during the development of a project, as there will be no need to worry about SQL dumps, as the database is always versioned, and sent to a Git repository.

4.3.3 CRUD and models

After the proper configuration of the application, the developer will be able to access the Gii tool. Gii is a CRUD and model generator for the Yii2 framework. A model in Yii2 is the representation of the table in code form. By default, using the default configuration that Yii2 comes with out of the box, the tool can be found with the URL http://localhost/crawler/index.php?r=gii [20].

After much usage of Yii2 framework, I decided that it would be better to split the models into two parts: a base class that will be used during model generation for easy future variable insertion, should it be required, and the actual class where all the actual functionality be done. Because the former model class always should extend an ActiveRecord class provided by Yii2, or any child derived from the said ActiveRecord class, this means that the latter class will also have the ability to override the inherited classes. The most important inherited methods are related to the save workflow of ActiveRecord class such as beforeSave and afterSave. This is where some of the scraping will have to be done.

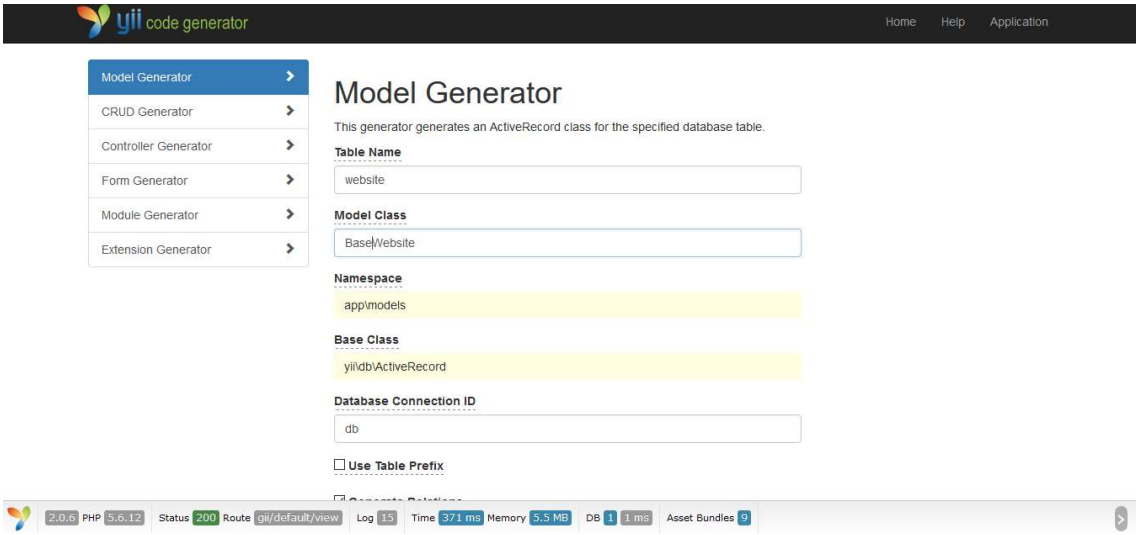

Figure 3: Example of the usage of the Gii module [21].

Figure 3 shows the user interface of the Gii module in Yii2. This particular example is where all the details of a model are given such as the table name, which will be used for attribute generation. After generating the models, one can also generate a CRUD. This will also generate the controller which will act as an entrance point to our application. Once the CRUD has been created, we can test the entrance point by going to the URL http://localhost/crawler/index.php?r=website.

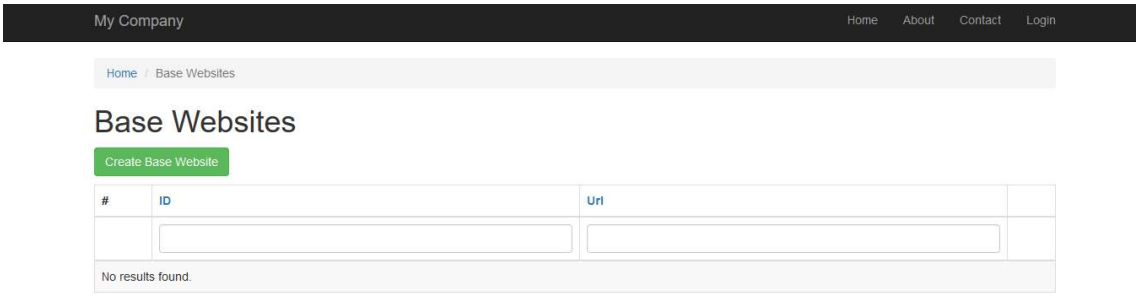

Figure 4: What the URL http://localhost/crawler/index.php?r=website looks like.

Figure 4 demonstrates that the application has been successfully created with the CRUD and the database tables. This means that the development of the logic behind the web crawler can begin.

4.4 Selecting an HTML parsing tool

PHP has many native as well as third party HTML parsing tools and libraries. Choosing the right one will determine how easy the process of implementing the scraping tool will be. Another factor that will be affected by the choice of the parsing tool, will be the speed at which each page will be parsed and stored. This means that an effective library must be chosen for the task of parsing the retrieved content. If a library that is slow is chosen, the length of time that the application takes to parse a website will take significantly longer than a library that is effective at its job.

When looking for an HTML DOM parser, the main criteria was that it could support jQuery-like selectors. At first, PHP Simple HTML DOM Parser seemed like the tool for the job, but after doing some research, a benchmark was found which revealed that this library was slower than the native implementation, which has jQuery-like selectors built in. [22.] For that reason, the native implementation was selected.

While this implementation does not have true jQuery-like selectors, it comes quite close to it. This library uses XPath selectors, which are, in reality a way of selecting nodes in XML. Despite this fact, and XML being closely linked to HTML, these selectors will work for getting elements of HTML needed for the accurate parsing of the data needed.

5 Technical implementation

5.1 Basic implementation

Because of the recursive nature of the web crawler, I decided that the best approach was to create a recursive architecture for traversing pages. The recursive method will be found in the Page model. The recursive entry point shall be in the Website model in the overridden method afterSave. This entry point was chosen because the Page models that shall be generated will need a foreign key to be defined.

Once the application enters the recursion entry point, the web crawler will have to get the first web page. Most of the time this will be the initial request or console command. Both of these options will have a parameter denoting the first page to be processed. In theory, any page of a website can be given, and the web scraper shall find its way to the index page of the website through the navigation bar link. This will be done with the PHP binding for the library libcurl. This library handles all requests, as well as retrieves the HTTP status code as well as gets the data, in this case an HTML document.

However, there is a chance that, due to a human error, either by the user of the web scraper or by the developer leaving out <a> tags for links, a page with no links to the index page might be given. Should this be the case, only one page shall be recorded.

Generally, down the line of recursion, a 404 (not found, or generally any 4xx error) or any of the redirect statuses can be returned by the request (and possibly, but very unlikely, any 5xx errors). As such the function that I shall be using recursively shall have to have a handler for these possible outcomes. Since the most 3xx statuses are redirects of one sort or another, these will have to be handled differently from the 4xx and 5xx statuses. The former statuses will need to redirect until a 200 or any of the aforementioned errors are encountered. However, there is a possibility that an infinite redirect loop might transpire. Therefore, a maximum redirect attempt constant must be set. Once this limit is reached, the redirect, shall be treated as an error. The 5xx status code family shall not be logged into the database. Neither should those pages that reach the redirect maximum, since they are technically pages that do not work. This is because the specification of the application says clearly that a count of all the pages that are returning HTML be counted.

During this traversal the delay after each request as specified by the web scraping rules in section 4.1 must be applied. It was decided that the acceptable delay between each request would be two seconds. While this does increase the execution time, it is a necessary precaution, as we do not want to crash the web server the customer's web server, nor attract unnecessary attention to the web crawler.

Should, however, the request return a status code of 200, the real work of parsing can begin. The selected tool uses xpath selectors, which are very close to jQuery and CSS selectors. Using these selectors, all the links can be selected, as well any and all tags which have the possibility to give useful information about the website.

6.2 Caching

One thing that came up during the development of the application is the need to cache the web pages. The reason for this is if, for example, one of the web pages encounters a problem, such a 500 status error, the page is left out of the evaluation. The end user might notice this error if they go through the web page. Should the end user decide to notify a potential customer about the problem, and the potential customer decide to fix it, a revaluation will be required. There is no need to re-fetch pages that have already been evaluated for a certain time. Not only does caching of web pages speed up the re-evaluation of a website, but it also lowers the potential request load on the customer's web server. The caching was done through checking whether a certain time period has passed from the last evaluation. If the allotted time has not passed, the cached version of a page will be loaded, instead of requesting a copy of the HTML content from the server.

The cached copies of the HTML files will be stored in a predefined place on the server where the application will be run. This folder should not be web accessible. This is because it is not a good practice to have other people's data available to be seen on someone else's server. The actual file was stored by using a PHP function file_put_contents. This function handles all of the handle opening and closing that is usually related to writing a file, effectively reducing the code required to do the operation to one line.

I decided that the name of the cached file would be based on the database data. Each file will have the website id followed by a dash followed by the page id. This ensures that each file name is unique and no caching clashes shall happen.

6.3 HTML parsing

The most important thing for a web scraping application to know is where to go after the initial page. This is why this application must get the links the initial page links to. Luckily the chosen library had a very easy way of getting all of the links from an HTML document.

```
DOMDocument::getElementsByTagName('a');
```

Listing 7: The method used to get all of the links in an HTML document

Listing 7 shows how the fetching of all of the links on a page happens. The return type is an array, so the result can be easily fed into the foreach loop. This loop takes care that every link on a page is fed into a Page model. Before the Page model is further processed, the link is validated to make sure that it is not an external page, an email or a file. Each link is subjected to a uniqueness validation as well, as there is no need for duplicate entries in the database. Allowing duplicate database entries would also increase the execution time of the application, further increasing the load on the server that is being scraped. Once a link has been sufficiently validated, the process outlined in section 5.1 can take place again for the said link, as well as any of the following descriptions, as this process is, in a sense, a recursive process.

```
$xpath = new \DOMXPath($this->_dom);
$nodes = $xpath->query('//body//text()');
$text = '';
foreach($nodes as $node) {
    if($node->nodeName === '#cdata-section') continue;
    $text .= " $node->nodeValue";
}
$text=str_replace('ä','a',$text);
$text=str_replace('ö','o',$text);
$text=str_replace('Ä','a',$text);
$text=str_replace('Ö','o',$text);
$text=str_replace('å','o',$text);
$text=str_replace('Å','o',$text);
$this->word_count = str_word_count($text);

$nodes = $xpath->query('//body//img');
$this->image_count = $nodes->length;

$nodes = $xpath->query('//body//p');
$this->paragraph_count = $nodes->length;
$this->save(false);
```

Listing 8: Data fetching from the HTML document

Once the link validation process succeeds, the process of extracting useful data about the document behind the link can begin. Listing 8 is the actual code of getting all of the data as specified by the project manager. The variable $this is the Page object as called from inside the class itself. The "//body//text()" selector was used to get any textual data from the page. This selector excluded all of the HTML tags. After inspecting the output of each of the nodes returned by this selector, it was noticed that, on top of all of the textual data of the document being returned, all of the JavaScript code present on a page was also being selected. Luckily, the JavaScript node name was different from the actual textual data, so it was set to be ignored using the continue statement in the foreach loop. Following this operation, I tried to use the PHP function str_word_count, but quickly found out that it was not safe to use it with umlaut characters. If umlaut characters were to be given to the function, a word would be considered terminated at an umlaut character, and therefore the word count was incorrect. The solution to this problem was to replace the said characters with regular characters using str_replace. Doing this does not have any consequence, as the word count is being stored, not the actual textual data.

The final part of the specification required the image count and the paragraph count. This was easier than the previous part as the used XPath selectors accurately gave the required values. After all of the values are calculated, the model can be saved. The

false boolean value given to the save function denotes not needing to validate the data being saved, as it is, by this point, already validated.

6.4 Dealing with long execution times of the application

One thing that was apparent from the very beginning was that the execution time of the application would not be trivial. This is in part due to the rule web scraping rule stating to be nice to the target server, and in part due to the data fetching functions. Even during the development of the application, it became apparent that having the long execution times will not be friendly for the end user of the web application, as the end user will probably want to do something else with their browser while the scraping function is being run.

The solution to this problem was an asynchronous pseudo multithreading operation. PHP, however, does not support natively multithreading. This is where the pseudo part comes into play. What is meant by this is when a web request is received by the server, it triggers a PHP console application. Because the Yii2 framework has a console interface natively, achieving a console call, and using the already written code in the web scraping operation using PHP's native console calls was easy. However not waiting for the console application to end was trickier.

```
$cmd = yii::getAlias('@app') . "/yii scrape/index ";
if (substr(php_uname(), 0, 7) == "Windows"){
    pclose(popen("start /B ". $cmd, "r"));
}
else {
    exec($cmd . " > /dev/null &");
}
```

Listing 9: Invoking a console application in both Windows and UNIX systems. [23.]

Listing 9 demonstrates how to call a console command in both Windows and Unix environments and not wait for the command to finish execution. This is done through the use of environment specific functions provided by PHP This allows the end user to do other stuff on the web application, and not have to wait for the application to end. This should be called on an evaluation request, either through AJAX or a regular web request.

6.5 Modularization

The final step of the implementation is to modularize the code of this project. The reason behind this is the ease of use of this code in other projects. Another added benefit of modularization is that Composer can be used to install the module into the project. Once the module is installed, and if the module itself is properly configured, the code can be called directly through a web request. In the case of this implementation, this feature will be very useful, as the programmer who includes this module does not need to implement any code in the application that the developer is developing. All the developer needs to do is send a web request with the right route. The first step in modularization is to move all of the code to a separate folder.

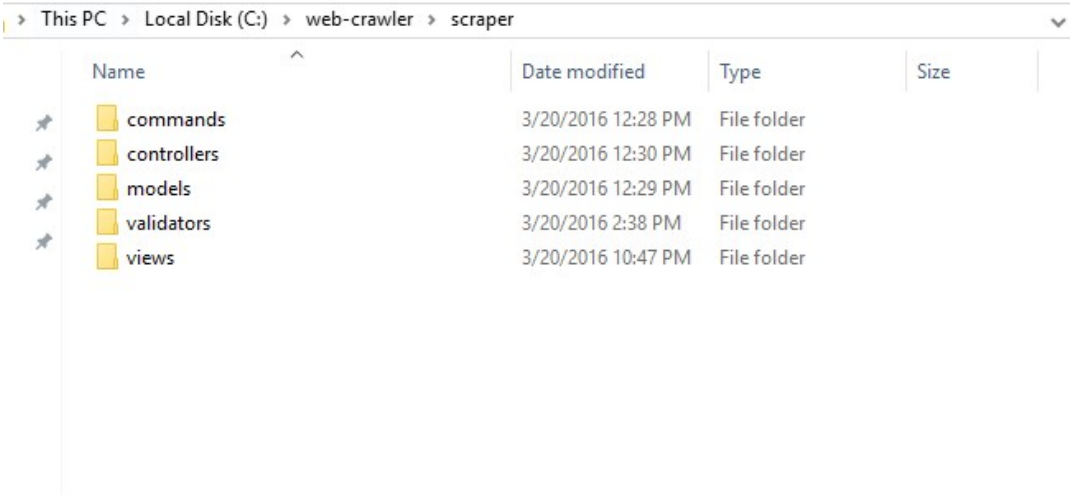

Figure 5: The initial structure of the module folder after the transfer

As figure 5 shows, only the relevant folders to the module, and the files that are inside them, were transferred. The next step is to create a Module class that is provided by the Yii2. In this Module class, we will define everything that the module needs to work, such as the controllers, and where to find them. This class also controls what the route to these classes will be called through. This class may, should the developer so desire, rename the controllers as need be through the controller map. This also negates the requirement to set the controller namespace, but if the module has both CLI and web application controllers, this option becomes very inflexible, as the console routes will be inconfigurable.

```php
namespace pro5\webscraper;

use Yii;

class Module extends \yii\base\Module
{
    public function init()
    {
        parent::init();
        if (Yii::$app instanceof \yii\console\Application)
{
            $this->controllerNamespace = 'pro5\
webscraper\commands';
        } else {
            $this->controllerNamespace =
           'pro5\webscraper\controllers';
        }
    }
}
```

Listing 10: The setup of the module

Listing 10 demonstrates how to set up the module class for this project. Since this module has both console and web parts, it must ensured that the unrelated controllers leak to the wrong place. Because of this, a check must take place to determine which application was called. Based on the result of the check, the location of where the application will look for controllers will be set. This way we can mitigate any possible unauthorised actions to a minimum.

```php
$config = [
    ...
    'modules' => [
        'webscraper' => [
            'class' => 'pro5\webscraper\Module',
        ]
    ],
    'aliases' => ['@pro5/webscraper'=>'@app/webscraper'],
    ...
];
```

Listing 11: Additions to the configuration file to initialize the module

Another important part of setting up the modules is setting the namespace aliases in the configuration file. Listing 11 shows how this project set up the module. This way this application can now include the required models, e.g the Website model from the module using the namespace pro5\webscraper\models\Website. Setting up a module also open the controllers to be used from the web browser and conole. This configuration snippet must be included in both the web application and the console application. If, however, the advanced template was used, putting this configuration snipped in the common configuration will suffice. To test that all of the above modularization steps were successful, both the the Yii2 console application can becalled to get the list of all of the available commands, as well as call one of the website side controllers.

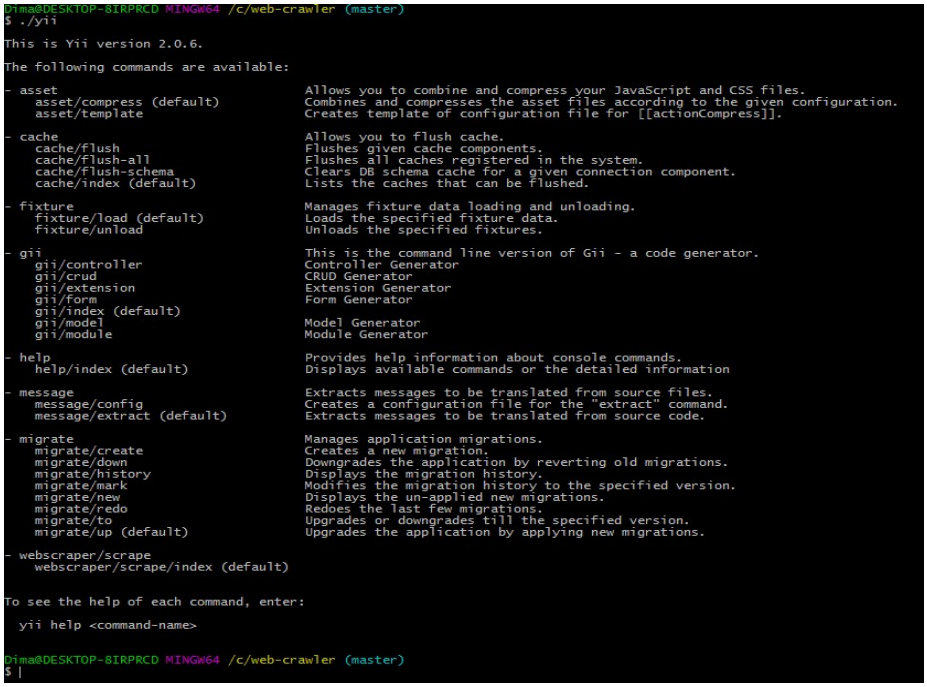

Figure 6: The controller in the Module that has just been created is now available.

Figure 6 demonstrates that the configuration was successful on the console side. This means that when the module is imported to another project through Composer, the command that has been created can be invoked using it through the module route, followed by the controller name and action name without the actual need of implementing the code itself. To test whether the web application was configured correctly, a browser must be used.

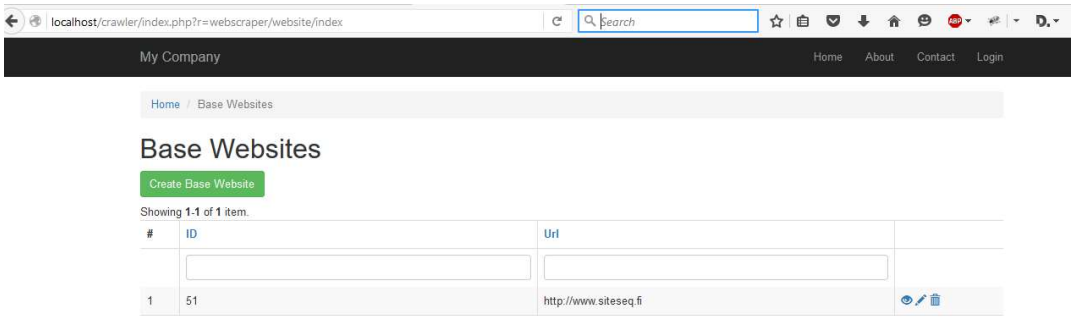

Figure 7: Data fetching using the webscraper module.

Figure 7 illustrates that the module has been configured correctly and the controller is being called from inside the module. These two images signify that the module has been set up correctly, and the module is ready to be integrated into other projects when needed.

Once all of this is set up, and working, the final step is to set up a compose.json file in the module. This file acts as a meta data storage, as well as holds information on dependencies of the module. This file is used by Composer to identify the module, as well as get any dependencies that are stated in this file.

After that is done, the module can be pushed to a Git repository to be included in other projects. An include of this module happens by putting the user name followed by a forward slash followed by the project, should it be in a public GitHub repository. However, here the code shall be on a private server with a private Git server. In this case the previous still holds, but the developer must also use a Composer variable called repositories. Once the repository location as specified by the Git web GUI is added, the Composer application will be able to find and download the code associated with it.

6 Benchmarking

6.1 Traversing

To test the functionality of the application, I used ten random websites from Seq 5 Oy's lead database. The initial results for the traversing part were favorable, as most of the sites tested went through the traversal part without problems. Some sites, however, had unfavorable behaviors. One such site had links with their own domain name in the links. While a safety net of sorts for this scenario was developed, the initial implementation did not handle well these types of links. As a result, the script created unnecessary rows in the database. This issue was fixed by removing the domain name, should it have been the same as the domain that was used to enter the application, as the application was built around the internal URLs.

Another problem that was encountered, although not as big as the former, wass when a website did not hide the script name. This means that the website was showing index.php and the page was given through a GET parameter. Although not a problem with sites that use pure HTML code, because each site has a unique page, it is a problem when sites use PHP application frameworks, and do not hide the script name. The problem here is mainly with the fact that the home page has the risk of being counted twice. This was the case with one of the test subjects. A way to fix this would be to ignore pages that have just index in them, and nothing else.

6.2 Data gathering

Because the traversal algorithm worked almost flawlessly, the page count was, in most cases, accurate. The previously mentioned outliers were fixed and should not be a problem in the future. However, the taken sample was too small to be certain of whether the page count will be accurate for all pages, as there will be some certain cases when some data is either in a secure area, or the links differ from the norm.

The gathering of image and paragraph counts was mostly accurate. There is nonetheless an error margin that was noticed in the image count. This was in the case when an image was no longer existing for one reason or another on the server. This, however,

was noticed on one of the websites tested, and only one image on the whole website. This may be an outlier, but it must be taken into account when searching for images.

The word count, on the other hand, was trickier to get right. The reason behind this was the algorithm used. On average it returned 10% more words than there actually were. One of the main reasons for this is the fact that it is hard to define to a computer what counts as a word. Also the algorithm used could not handle UTF-8 encoding properly, and if a character that was encountered has an accent, the algorithm would terminate the word at the character. Another problem that was noticed was the general formatting of the page. Some of the pages had strange parts that anyway broke the word, and the algorithm counted it as a word anyway. However, after consulting with the project director at Seq 5 Oy, it was decided that, since the word count would not impact the price that much, a 10% error on average would be acceptable.

6.3 Execution time

As previously discussed, the execution time of the application will be extended by two seconds per page. That means that a website with ten pages has an expected wait period of 20 seconds. On average a page was analyzed in 45 milliseconds, which makes the theoretical execution time stand at 20.5 seconds for this example page. However, the transfer of data is not taken into account. If the internet connection on either end is under a heavy load by some other operation, or the connection has a low bandwidth, the data transfer speed may suffer, thus extending the execution time beyond this theoretical execution time.

The tests that were run on websites of different sizes reflected this. Websites with 22 web pages that were expected to have a 45 second execution time, generally had an execution time variation between 60 seconds up to 75 on a connection with a speed of 2 Mbit/s (the speed of a mobile network connection). This reflects that the bandwidth of the internet connection that has the potential to fluctuate a lot really does impact the execution time of the application. However, these extreme conditions might never happen in the wild as the real connection speeds that web servers provide are much greater and are generally more stable than a mobile network, which can fluctuate greatly in a short period of time.

Furthermore, when tested on a 50 Mbit/s connection, it was discovered that the real time was 55 seconds in the case of a 22-page website. Upon further investigation of the code, it was discovered that the reason behind the extra time taken for the operation to complete was the creation of the connection every time it was needed. The validation rule of the URL is also to blame for this slowdown. However, since the application shall always run in the background, this slowdown does not, in theory, really matter.

7 Conclusion

This thesis focused on how to create a web scraping application using the Yii2 framework, and the tools that the framework provided, as well as the use of PHP tools that were available. The main intention was to create a tool that automated website crawling and retrieved useful information from a website. This goal was met, and a robust application was created during the writing of the thesis.

The success lies in the website traversing part, as it was the main part of the project. The traversal was made as robust as possible, while following the web scraping rules. With the testing that was done on the system, all of the initially obvious problems were fixed, which made the system ready for rolling out on the spot. However, the data fetching was less successful due to the limitations of the algorithms used. These inaccuracies can be fixed, should further development of the system take place.

As a side effect of witing this thesis, I also helped streamline the development process of applications in Seq 5 Oy. The main problem internally was the underutilization of the migration tools provided by the Yii2 framework. With the help of the research done for this project I was able to implement internal migration tools for Seq 5 Oy, and disposed of the need to have the database structure transported over database dumps in favor of using the migration tools.

After completing this project, I feel that not only will this system save time of the the sales team, but the research done for the project will benefit the company in a broader sence. This thesis contains many a time saving insight, one of which has already been implemented and is in use inside the company.

References

1. Lerdorf, Rasmus. PHP on Hormones – History of PHP Presentation by Rasmus Lerdorf Given at the MySQL Conference in Santa Clara, California [online]. April 2007.
   URL:
   http://web.archive.org/web/20130729204354id_/http://itc.conversationsnetwork.org/shows/detail3298.html . Accessed 5 April 2015.

2. History of PHP [online]. URL: http://php.net/manual/en/history.php.php. Accessed 5 April 2015.

3. Trachtenberg, Adam. Why PHP 5 Rocks! [online]. July 2004.
   URL: http://www.onlamp.com/pub/a/php/2004/07/15/UpgradePHP5.html. Accessed 7 April 2015.

4. Summers, Ed. Paying Homage to Perl (PHP) [online]. February 2003.
   URL: http://www.theperlreview.com/articles/php.html. Accessed 5 April 15.

5. PHP History. [online]. NuSphere.
   URL: www.nusphere.com/php/php_history.htm. Accessed 5 May 2015.

6. PHP: todo: php70. [online].
   URL: https://wiki.php.net/todo/php70#timetable. Accessed 7 May 2016.

7. PHP: New Features. [online].
   URL: http://php.net/manual/en/migration70.new-features.php. Accessed 7 May 2016.

8. Taei, Pejman. 10 Advantages of PHP over Other Languages [online]. February 2013.
   URL: http://www.webnethosting.net/10-advantages-of-php-over-other-languages/. Accessed 1 March 2016.

9. Pavlovskaya, Natalie. Top 10 World Brands Using Magento [online]. November 2012.
URL: http://blog.mageworx.com/2012/11/top-10-world-brands-on-magento/
Accessed 1 March 2016.

10. Lerdorf, Rasmus. php.internals: Re: Function Name Consistency [online]. January 2014.
URL: http://news.php.net/php.internals/70950. Accessed 5 April 2015.

11. Building Your Website with PHP (Advantages and Disadvantages) [online]. November 2015.
URL:http://webhostingmedia.net/building-website-with-php/.
Accessed 28 August 2016.

12. O'Dell, Jolie. 8 Experts Break Down the Pros and Cons of Coding with PHP. [online]. November 2010.
URL:http://mashable.com/2010/11/19/pros-cons-php/#DZMXk6ANKOqk. Accessed 29 February 2016.

13. Skvorc, Bruno. 3 Ways to Develop Cross Platform Desktop Apps with PHP [online]. December 2014.
URL:http://www.sitepoint.com/3-ways-develop-cross-platform-desktop-apps-php/. Accessed 29 February 2016.

14. Reda, Greg. Web Scraping 101 with Python [online]. March 2013.
URL: http://www.gregreda.com/2013/03/03/web-scraping-101-with-python/. Accessed 3 August 2015.

15. Mischel, Jim. Why Is This Regular Expression So Slow? [online]. June 2008.
URL: http://www.informit.com/blogs/blog.aspx?uk=Why-is-this-regular-expression-so-slow. Accessed 11 August 2015.

16. Manually Install Laravel/Composer Packages? [online]. March 2015.
URL:
http://laravel.io/forum/03-29-2015-manually-install-laravelcomposer-packages.
Accessed 27 January 2016.

17. Git. [online].
URL: https://git-scm.com/. Accessed 28 January 2016.

18. Getting Started - The Command Line [online].
URL: https://git-scm.com/book/en/v2/Getting-Started-The-Command-Line. Accessed 7 May 2016.

19. Yii Software LLC. Yii Framework [online].
URL: http://www.yiiframework.com/download/. Accessed 27 January 2016.

20. Yii Software LLC. Gii Extension for Yii 2 [online].
URL:http://www.yiiframework.com/doc-2.0/ext-gii-index.html. Accessed 27 January 2016.

21. Yii2 Framework, Gii Code Generator [computer program]. Yii Software LLC; December 2013

22. Whitlock, Byron. Simple HTML DOM Is Slow [online].
URL: http://byronwhitlock.com/FastCrawl/benchmark.php. Accessed 27 January 2016.

23. van den Brink, Arno. PHP: exec – Manual [online]. October 2008.
URL: http://php.net/manual/en/function.exec.php#86329 Accessed 7 May 2016.