Huy Phan

Building Application Powered by Web Scraping

Helsinki Metropolia University of Applied Sciences

Information Technology

Bachelor Thesis

01 March 2019

PREFACE

Writing the thesis while working is a tall order. I am thankful for my family, especially my mom, who were so patient and gave me amazing encouragement to complete the paper. I want to thank my supportive colleagues at work who provided helpful technical advice and information during the process. I also appreciate the people who develop the Scrapy web framework, a wonderful Python library with good documentation and community support for Web Scraping.

Espoo, 06.03.2019

Huy Phan

Author(s) Title Number of Pages Date	Huy Phan Building Application Powered by Web Scraping 31 pages 01 March 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Janne Salonen

Being able to collect and process online contents to help can help businesses to make informed decisions. With the explosion of data available online this process cannot be practically accomplished with manual browsing but can be done with Web Scraping, an automated system that can collect just the necessary data. This paper examines the use of Web Scraper in building Web Applications, in order to identify the major advantages and challenges of web scraping. Two applications based on web scrapers are built to study how scraper can help developers retrieve and analyze data. One has a web scraper backend to fetch data from web stores as demanded. The other scraps and accumulates data over time.

A good web scraper requires very robust, multi-component architecture that is fault tolerant. The retrieval logic can be complicated since the data can be in different format. A typical application based on web scraper requires regular maintenance in order to function smoothly. Site owners may not want such a robot scraper to visit and extract data from their sites so it is important to check the site's policy before trying to scrap its contents.

It will be beneficial to look into ways to optimize the scraper traffic. The next step after data retrieval is to have a well-defined pipeline to process the raw data to get just the meaningful data that the developer intended to get.

Keywords	Scraper, Web, HTML, Python, JavaScript

Table of Contents

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
CAPTCHA	Completely Automated Public Turing Test To Tell Computers and Humans Apart
CSS	Cascading Style Sheets
CSSOM	Cascading Style Sheets Object Model
CSV	Comma Separated Values
DO	Document Object Model
GDPR	General Data Protection Regulation
HTML	Hypertext Markup Language
PDF	Portable Document Format
REST	Representational State Transfer
SQS	Simple Queue Service
UI	User Interface
URL	Uniform Resource Locator
XML	Extensible Markup Language
RFDa	Resource Description Framework
JSON-LD	JavaScript Object Notation for Linked Data

1. Introduction

Information is one of the most valuable assets that can be found anywhere on the internet. Many web services and applications require a large amount of data to work properly. Examples of such applications are web search engines (Google, Duckduckgo), product price and feature comparison (versus.com, pricealarm.net) or internal tools in many companies for market and competitor analysis. In order to extract information on the internet and turn them into usable formatted data, those applications usually leverage Web Scraping.

The best approach to understanding such a complex system as a web scraper is to get into the practice of making one. Therefore, the scope of this research is to study Web Scraping and the process to build a data-driven application with data provided by Web Scraping.

To understand this process, two applications will be created. The first application uses Web Scraping as a backend service and provides real-time product search results comparison between three popular online electronics stores in Finland. The second application is a simplified version of Google's web search engine, with data provided by running Web Scraping daily.

This thesis contains 5 chapters. This section covers chapter one. Chapter two provides theoretical backgrounds for Web Scraping Technology and Development as well as potential challenges. Chapter 3 documented a real-life application development using Web Scraping. Chapter 4 discusses the results of the work done, with some practical findings during the development process. And chapter 5 is the conclusion, where the results are compared to what was learned from the theoretical background.

2. Theoretical background

This section consists of eleven parts. The first section introduces Web Scraping. Then, the next two parts look into the overview of web application components and rendering process. Next, Web Scraping technologies are discussed. Then, Web scraping framework and architecture design are examined. Furthermore, the focus is on the analysis of some rules and legal aspect regarding Web Scraping. After that, Web Scraping Service, Scaling, and Community Support are studied. Last but not least, the section discusses the challenges and future of Web Scraping.

2.1 Introduction to Web Scraping

Web scraping is a process that automatically collects and extracts data from the internet. According to Mahto and Singh (2016: 689-693), web scraper is one of the most efficient tools to extract data from websites. The purpose of web scraping is to extract a large quantity of data and save to a local environment [9]. In the early days, the only way to extract data from websites was by copy-pasting what one saw on the website. Web scraping is becoming a popular technique as it allows new startups to quickly obtain large amounts of data. Most typical examples of web scraping are price comparison and review websites. Big corporations also use this technique to rank and index web pages.

Web scraping process involves crawling and parsing [9]. Crawlers go to the target URL, download the page content and follow links in the content according to developer's implementation. Parsers parse downloaded contents and extract data. In order to understand and implement parsers and crawlers, it is important to know the basic components of web applications and website rendering process.

2.2 Web application components

Websites can have different components and structures. However, in general, there are 4 components: HTML, CSS, JavaScript and multimedia resources. Additionally, Document Object Model, CSS selector and Xpath provide a way for developer to control the web content.

Hypertext Markup Language (HTML)

HTML is a markup language that defines the structure of a web application. HTML elements are the main components of a HTML page. [1]

Cascading Style Sheet (CSS)

CSS describes how an HTML document is presented to end users. [2]

JavaScript

JavaScript is a scripting language which is mostly used on the web. In most web applications, JavaScript makes it possible for users to interact with application components, validate input and make API call to the backend server. [3]

Multimedia ***Resources***

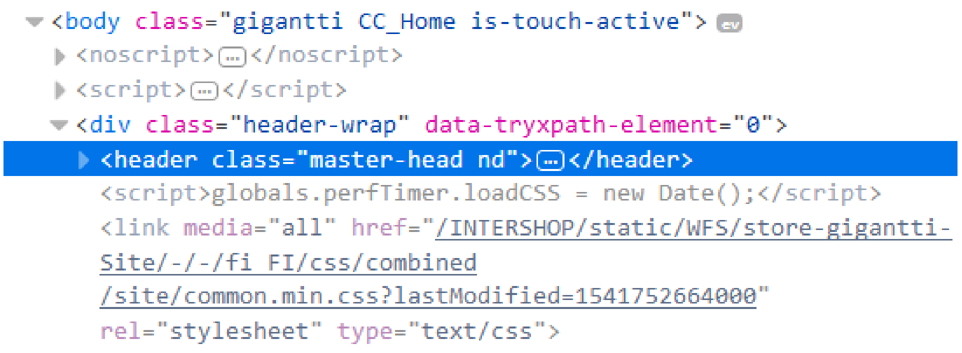
File, Video, and Image usually are embedded in the application as links. These links, especially for videos, are usually hidden by some form of client-side encryption or encoding.

Multimedia files are usually sensitive data that should not be downloaded and stored in Web Scraping database. However, storing the links to multimedia files might be acceptable because that is what a web browser does when the user visits web pages, and big companies such as Google also store them.

XPath and CSS selectors

Xpath is a powerful query language for fetching nodes from an XML document. Because HTML is a subset of XML, XPath can also be used to fetch an HTML element. [4]

Even though XPath is powerful, it is hard to read and write. The CSS selector is a more human-friendly alternative which allows selecting the HTML element based on the CSS styles, class name, or id associated with the element. [5]. Figure 1 shows an example HTML markup of a page in Gigantti website.



```
<body class="gigantti CC_Home is-touch-active">
  <noscript>…</noscript>
  <script>…</script>
  <div class="header-wrap" data-tryxpath-element="0">
    <header class="master-head nd">…</header>
    <script>globals.perfTimer.loadCSS = new Date();</script>
    <link media="all" href="/INTERSHOP/static/WFS/store-gigantti-Site/-/-/fi_FI/css/combined/site/common.min.css?lastModified=1541752664000" rel="stylesheet" type="text/css">
```

Figure 1. Simple HTML markup with CSS classes

As seen in Figure 1, to select the div element with header-wrap class name there are at least 2 ways to select in Xpath:

- *\/html/body/div[1]* : get the first div inside body inside html tag.
- *//div[contains(@class,'header-wrap')]* : get all div tags that has class header-wrap

Both return the same result in this case. According to several benchmark measurements from Elemental Selenium website, in old browsers, XPath selectors also give a faster performance on getting HTML Elements. However, in modern browsers, with smart optimization, getting HTML Element speed is almost the same for both XPath and CSS.[6]

With ”*div.header-wrap”* CSS selector, it is easier to understand. However, it is more difficult to create more advanced HTML Element selection criteria.

Document Object Model (DOM)

DOM is an interface for programming languages to access HTML documents. [7] It describes these documents as tree nodes and objects, which make it possible for programming languages such as JavaScript to select nodes using XPath or CSS selectors, to change the web page’s structure dynamically based on user interaction.

2.3 Web application rendering process

Even though web browser rendering knowledge is not required to scrape the web, it is important to understand on a high level how the browser renders web pages and how it interacts with any backend server.

There are several steps on how web applications render contents when users enter the page:

- Firstly, the server processes the request and returns the initial documents with HTML, JavaScript, CSS. Once static contents are downloaded, the web browser's rendering process begins. HTML and CSS are converted into DOM and CSSOM trees [8]. Figure 2. Shows an example web page DOM and CSSOM structures

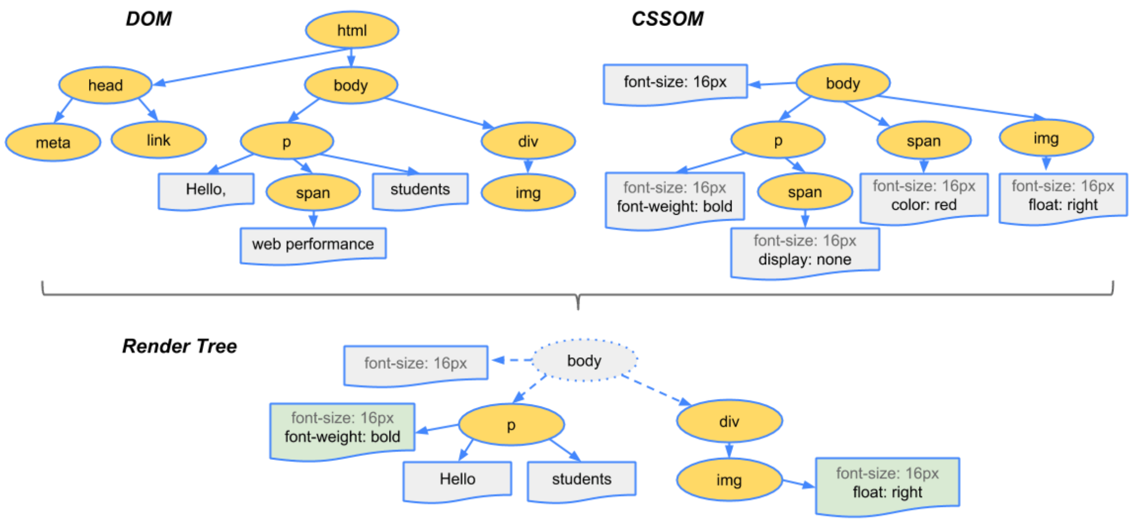


Figure 2. Diagram showing how HTML DOM combine with CSSOM into Render Tree [8]

During the rendering process, CSSOM and HTML DOM are combined into a single render tree. After that, the browser paints the browser view with boxes based on the render tree.

- Secondly, the browser reads and executes JavaScript in the document, causing changes to the HTML the document content accordingly
- Finally, when the user interacts with components on the web application, there are three scenarios:
    + The application goes back to step one for the server side to process another request with different parameters and render another page.

+ The application triggers a JavaScript function to change the web content. In this step, JavaScript might make a call to an API (Application Programming Interface) to get more data.
+ Both above-mentioned scenarios happen sequentially.

Depending on when targeted data for scraping are rendered or available from API call, different scraping techniques need to be applied. Sometimes, a real web browser is used for web scraping. More detail will be discussed in the next section.

2.4 Web Scraping technology

2.4.1 Web crawlers

In order to extract data from websites, a discovery process is required. Web crawler is a bot that visits websites and handle such process.
Mahto and Singh (2016) explain that a web crawler works in such a way that it loads a small list of links. Then it finds further links contained in those pages and adds them to a new list, called crawl frontier, for further exploration. The crawler needs to identify whether a URL is relative or absolute. For relative URL the crawler needs to identify the base of the URL [9]. A good crawler needs to be somewhat smart enough to detect circular references and slight variations of the same page in order to efficiently extract and store data.

2.4.2 Web Scraping parsers

In order to extract meaningful data from scraped data, Web Scrapers need to use parsers. They are usually implemented by programmers to format and extract specific detail from the data such as CV parser that can extract person name, contact information from an email content. Most Web Scraping libraries, has simple HTML parser support. Additionally, there are also parsers for special data stored as PDF, CSV, QR code or JSON. Real Web Browsers such as Firefox, Chrome have built in parsers. Web Scraping that is done by controlling real Web Browser can also use the browser built in parser.

2.4.3 Web Scraping policies

Mahto and Singh (2016) suggest that there are four main policies that a crawler needs to follow to act efficiently: selection, re-visit, politeness, parallelization.[9] By focusing on important links first the crawler can prune most of the unnecessary links and greatly reduce its search space. Sometimes pages are dynamic and the crawler needs to regularly check it for changes. A smart crawler would efficiently process just contents that had been altered. The crawler should also consider the effect it has on website performance and avoids causing heavy load on the site. And finally, to enhance the performance, the crawler utilizes parallel crawling where multiple sites are visited at the same time. Good inter-process communication is needed in order to avoid repeating the same work over and over again.

2.4.4 Web Scraping techniques

Upadhyay et al. (2017) outline the mechanism of web crawler in three steps: making a connection and learned of robot policies for automated operations on the site, extraction, filtering and processing of data, mapping the data into a structure that is useful for the intended application. [10]

Because of the way websites are structured, the most common technique to get relevant data is by utilizing the DOM of the page. This approach, however, has limited capabilities when dealing with dynamic pages with scripts constantly manipulating the content of the page. The authors found that for such pages a vision-based approach, where the crawler would render the page similar to how a human would read it, would allow the crawler to capture such dynamic contents. The third method is to use a combination of code and user interaction to visually zone in on interesting elements of the DOM. Such a method would require a tool that supports the desktop interface.

Lawson (2015) suggests that to avoid loading the page contents multiple times for different operations the crawler can cache the website somewhere when it first retrieved that page [11]. This helps a lot to reduce the wait time since accessing data from the local machine is a lot faster than downloading it over the internet. In order to do this, a technique called memorization is applied to the scraping function so that it first checks in memory for the page and only do the request if the page has not been loaded before. The URl is only downloaded if the cache for it is empty or the previous attempt to download it has failed. The author also proposes storing static contents on disk instead of memory so it can be loaded at a later time and not take too much resource. For disk storage, a compatible naming scheme for the URLs should be worked out since on most systems the filename cannot contain special characters that appear in the URLs. Some system also has a limit on the length of file names as well as the total number of files stored on disk. Since most modern websites have dynamic contents that mutate very often this cached data may become out of sync with the real online page contents after a while. The author recommends making the cache expired after some time period. The page is cached together with a timestamp so the program can determine whether its content is still valid. Another option for caching content is by using a database where it is easier to scale if the scraper has to deal with a large number of pages. Some database such as MongoDB has the ability to remove documents after a certain time interval so this works very well with the expiry of cache discussed above. Lawson also points out that using a database cache may be a bit less efficient than disk cache in case the set of pages is small [11]. However as the number of pages scales up the database approach can provide some advantage, notably the concurrency of data extraction.

Similar to Lawson, Haque and Singh (2015) also point out that the site itself can also alter its structure and content in such a way that is hard to extract useful information [12]. This technique can protect against most scrapers which rely on knowing some identifiable properties of the element in the DOM tree. The downside is that a frequent change in the website layout will confuse real users and discourage them from using the site. Another way to prevent scraper from easily extracting information is to present the information in an image or HTML5 canvas. However, the scraper can still use sophisticated OCR technique to extract text from images.

2.5 Web Scraping architecture

2.5.1 Scrapy framework

There are many web scraping frameworks written in different kinds of programming language. Among them, Scrapy is one of the most mature Web Scraping framework written in Python. Generally, most Web Scraping systems have similar architecture and components.

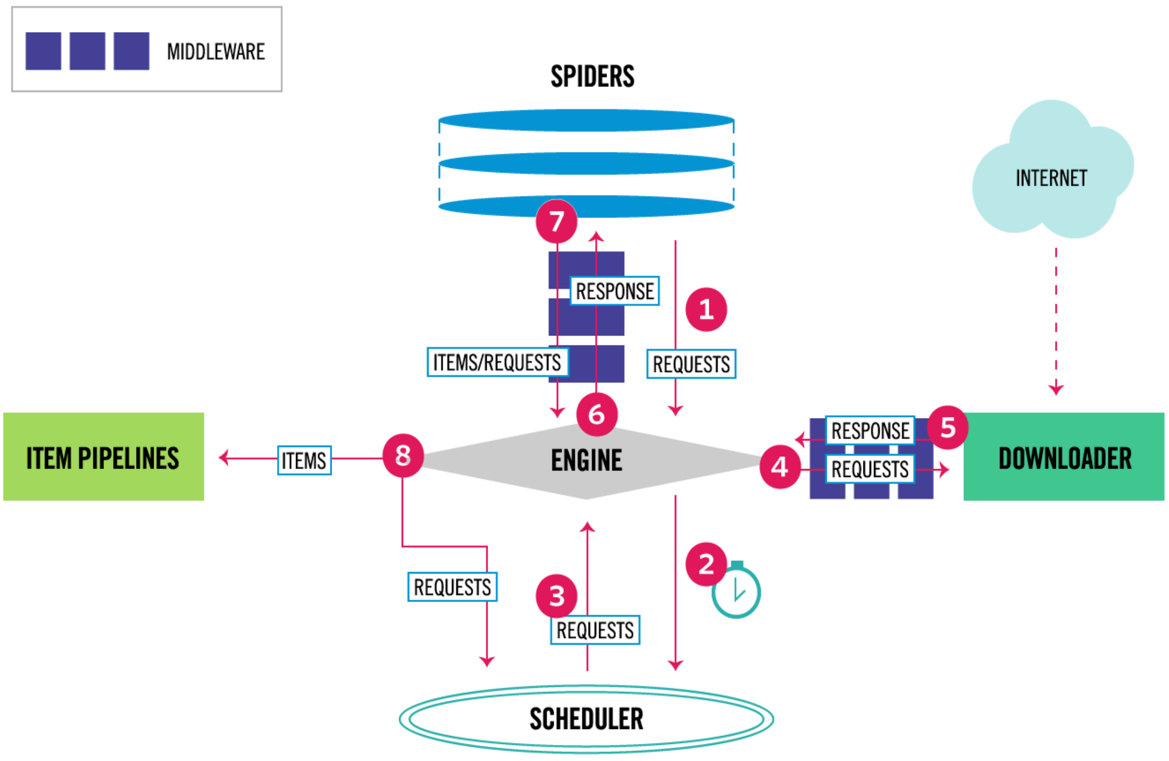


Figure 3. Scrapy architecture documentation [13]

Figure 3 introduces important components of Scrapy architecture. Each component is only responsible for specific task. Thus, writing tests is easier and scraper is more maintainable.

Engine

The engine is the center of Scrapy framework. It manages the flow between components of Scrapy. It is also responsible for listening and triggering events to react to certain actions such as request error, response error, exceptions. [13]

Scheduler

The Scheduler manages when a task should be run and it has a direct connection with task queues. It can regulate how much delay each request has. [13]

Downloader

The Downloader is where HTTP requests are made. It then stores and gives the HTTP response contents back to the engine in the normal case where real browser is not used. [13] In case that a real browser is used to make requests, a middleware which can control the browser will completely replace Downloader.

Spiders

Spiders are classes written by developers to define what actions the Scraper should do to get and parse specific web contents. [13] This is also where custom configurations for Downloader and its middlewares can be set. Parsed contents are then passed to Item Pipeline.

Item Pipeline

Item Pipeline process parsed data return by Spiders and do validation, custom transformation, cleaning and data persistence to a Data Storage such as Redis, MongoDB or Postgres. [13]

Downloader middlewares

Downloader middlewares intercept request and response sent to and from Downloader and enrich request and response data with custom metadata.[13] This is also where developers can add custom rules such as retry.

Spider middlewares

Spider middlewares function similarly to Downloader middlewares, but they stay between Spider and Engine.[13] Custom processing for Spider input such as changing target URL parameters can be done here.

2.5.2 Selenium Web Driver

Selenium is a library that exposes interface to control real Web Browser automatically [11]. As mentioned in Web application rendering process section, there are several way a website can load and render its content. Most server side rendered websites can be scraped by making simple HTTP request and use some XML parsers.
There are some websites that heavily relied on JavaScript and can only be scraped with real browser. Chaulagain et al. (2017) built their scrapers with Selenium. [16].
Even though Selenium makes it possible to scrape some complicated websites, it does

cost a lot more computer resources. Chaulagain et al. (2017) had to use scalable cloud services to actually support Selenium in their Scraper. [16] Selenium also slows down the Scraping process considerably because it needs to open the browser and loads the entire web page. The strategy is to try scraping without Selenium first and only fallback to it if there is no alternative solution.

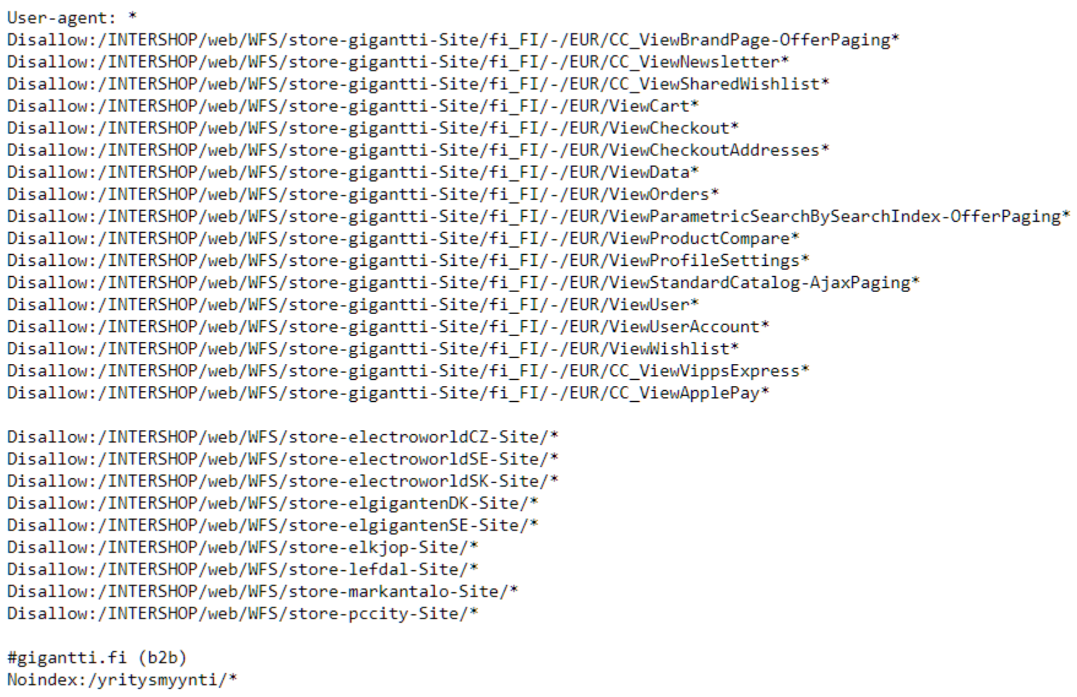
2.6 Web Scraping rulebook

2.6.1 Understand what, why and how data is needed

Before doing any Web Scraping, it is essential to know what kind of data is needed and for what purpose. There are scraping datasets publicly available online that are free to use and access. Next, it is important to read carefully term of service of the website that Web Scraping process will be conducted.

2.6.2 Respect Robot Exclusion Standard

Robot Exclusion Standard describes rules that bots should follow when accessing the website. Each website can have different rules. Those are defined in robots.txt in the root path of a website domain. [14] However, Web Scrapers might still crawl and extract data. This action is not recommended and will lead to IP banning and further serious lawsuits. In order to detect violations of robots.txt, website owners need to have proper logging, monitoring and alerts.
An example robot file found at https://www.gigantti.fi/robots.txt is showed in Figure 4



```
User-agent: *
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/CC_ViewBrandPage-OfferPaging*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/CC_ViewNewsletter*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/CC_ViewSharedWishlist*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewCart*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewCheckout*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewCheckoutAddresses*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewData*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewOrders*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewParametricSearchBySearchIndex-OfferPaging*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewProductCompare*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewProfileSettings*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewStandardCatalog-AjaxPaging*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewUser*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewUserAccount*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/ViewWishlist*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/CC_ViewVippsExpress*
Disallow:/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/EUR/CC_ViewApplePay*

Disallow:/INTERSHOP/web/WFS/store-electroworldCZ-Site/*
Disallow:/INTERSHOP/web/WFS/store-electroworldSE-Site/*
Disallow:/INTERSHOP/web/WFS/store-electroworldSK-Site/*
Disallow:/INTERSHOP/web/WFS/store-elgigantenDK-Site/*
Disallow:/INTERSHOP/web/WFS/store-elgigantenSE-Site/*
Disallow:/INTERSHOP/web/WFS/store-elkjop-Site/*
Disallow:/INTERSHOP/web/WFS/store-lefdal-Site/*
Disallow:/INTERSHOP/web/WFS/store-markantalo-Site/*
Disallow:/INTERSHOP/web/WFS/store-pccity-Site/*

#gigantti.fi (b2b)
Noindex:/yritysmyynti/*
```

Figure 4. Gigantti robots.txt file

Figure 4 shows that Gigantti does not allow scraping some of their pages such as ViewBrandPage, View Cart etc. However /search page is not “Disallowed” so it is still possible to scrape Gigantti search results.

2.6.3 Avoid scraping website with authentication

Authentication protects website and data from unauthorized access. It is not directly a mechanism to prevent Web Scraping but Web Scraper will need to implement login functionality with a real user account to extract data.

Usually, websites with authentication data are not available for public use. General purpose full-text search engines Web Scraper such as those from Google, Microsoft Bing usually exclude those websites from scraping. However, a person can still scrape his/her own personal data on that website if there is no other way to extract it. For example, scraping the list of his/her own personal favorite books or movies.

2.6.4 Detect anti-bot mechanism and stop

According to Haque and Singh (2015), web scraping uses automated scripts to collect a large quantity of data that can be then used for some applications. Some website owners, however, are concerned that web scraper poses a security threat and is intrusive into their business advantages. To protect against such threat, businesses are interested in building complicated anti-scraping tools.[12] Another reason that causes concern against web scraping is the limited resources of the website infrastructure. It is often hard to determine whether a connection is established by a bot or a real human. The authors explore the challenges that an anti-scraping system has to face in order to effectively mitigate such threats, without a severe impact on the user's experience.

Haque and Singh (2015) emphasize the need to allocate data related operations to suitable times when the server has a relatively low load. They suggest the use of IP classifications into three categories in order to combat scrapers: blacklist, graylist, and whitelist. For the graylist, which is those sources with unusual activities, there would need to be mechanisms such as CAPTCHA to verify that it is indeed genuine traffic. The authors also recommend using the robots.txt file to handle traffic from search engine bots. If a bot does not conform with the site's robots.txt file, its IP will be blacklisted. Another popular method, called browser fingerprinting, can also be applied, thanks to the fact that most common browser provides much information on its identities, while scrapers often lack this. To track the frequency of visits a separate table could be used to store data of hits per IP. When an IP seems to be the source of significant traffic compared to an average user it can be added to graylist. [12]

Once a website has blocked a web scraper, it is recommended to stop the scraping process immediately and try to understand what is the problem. Usually, there are some automatic service protection mechanisms take place as mentioned by Haque and Singh (2015). Web scraping might be allowed to continue if scraping speed and the number of concurrent requests is reduced. However, it is always best to contact the website owner and ask for permission.

2.7 Legal issues

Web scraping itself is not illegal. However, it can easily turn illegal if scraping is done without understanding the process.

Heydt (2014) claims there are two legal issues with Web Scraping, explained below: [14]

Data ownership: Essentially, whenever a person visits a website, his/her browser scrape the website and download the data to a local computer. It is similar to how Web Scraping works. However, downloading the data does not grant ownership of the data. Most of the time, website users are allowed to read, use and even share the data. It might be an issue when that scraped data is used to serve and gain profit from other people. In such a case, the term of service of each website should be read and reviewed by lawyers.

Denial of service: Making too many requests to a website might cause its server to crash, or stall the responses to legitimate users. This is also known as a denial of service. Thus, scraping too intensively will be a problem.

Depending on the website and data these issues can be mitigated. However, there is one legal issue that Heydt (2014) did not mention. It is the scraping of people personal data. Recently, Europe introduced the GDPR law regarding the use of personal data online. It is against the law to process and store people personal data. Personal data is the data that can be used to identify people. A simple example of personal data are emails, name, phone number, gender, id, age. Those are data that can be used directly to identify a person. However, indirect data such as anonymized favorite movies without real username is also considered personal data. Recently, Netflix released an anonymous user favorite movies dataset. Even though the user cannot be identified directly from this dataset, the University of Texas has published an article proving the real user can still be discovered by matching with IMDB favorite movies database.[15]

2.8 Web Scraping as a Service

Web Scraping scripts can be implemented and run locally. However, in real-world applications, Web Scraping works in the background with several services and applications to serve certain use case such as running scraper daily or on certain events. Thus it is common to build Web Scraping as a Service. There are several companies such as ScrapingHub.com, WebRobots.io who provide service that only require users to upload scraping configuration or code.

One of the challenge to build software as a service is to keep the development environment the same as the production environment. Developers might use machines with different operating systems which require different dependencies. Docker is a software created to solve this problem.

According to Heydt (2014), in order to run web scraping as a service easily locally and on the cloud, the scraping script needs to be exposed through a RESTful backend and package into a Docker container image [14]. A Docker container image is a software package which contains all the dependencies, runtime, system tools that can be executed on any machine with any operating system that has Docker support. Docker also makes it easier to scale Web Scraping service which will be discussed in the next section.

2.9 Scaling Web Scraping

Scaling Web Scraping require deploying multiple instances of Web Scraper into a cluster. While such a thing is difficult to do with traditional in house hosted servers, it is quite simple to do with cloud services from Amazon and Google as long as Web Scraping service is packaged with Docker.

To utilize the power of cloud computing, Chaulagain et al. (2017) decide to put all their scraping resources on AWS. The authors acknowledge that for web scraping to efficiently work with dynamic and interactive web pages, it needs to simulate human behaviors as much as possible, an assertion that was also mentioned by Upadhyay S. et al (2017). The authors find that with resources from scalable cloud platform it is easier to run scraper in parallel. And the resources could be scaled as needed should the load become too heavy. [16] In addition to getting massive data in parallel with cloud-based resources Chaulagain et al. (2017) also want to develop a system of distributed data processing with the same resources. [16] The advantage of using cloud service is that the scraper can operate in such a way that it is able to scrape multiple URLs at the same time and occupy just as much resource as needed. The systems use services from Amazon such as S3 and SQS to schedule tasks, store retrieved contents, and spawn more resources for the job. [16]
First, the input receives the target URL to be scraped and other configurations that tell which operations need to be done just as a user would browse the page. The job is put to an SQS queue with all relevant data for the scraper manager to read. After making sure that the configuration is valid the scraper manager will call Selenium Renderer to

process this job. Because Selenium has a real browser engine that is able to evaluate Javascript, it can go as far as a user could. After reaching the final page, a python library is used to transform its content into a DOM. After some stages of filtering the desired contents are extracted from the DOM and stored in DynamoDB. Due to the fact that all services including S3 SQS, DynamoDB are running in AWS cloud infrastructure, resources are optimized and scalable.

2.10 Community support for Web Scraping

Schema.org is a community that tries to build structure schemas for websites to implement so Web Crawler and Scraper could easily index and parse relevant data.[17] It would bring benefits to both website owner and web scraper. For example, e-commerce websites can directly show their “Adidas” products with price and all detail in Google Search engine when people search for “adidas” keyword. Furthermore, websites implemented this community schema usually has their result with a higher rank in search results. Web scraper and crawler will need less resource, show more up to date data and make fewer requests to websites in order to index them. Popular websites such as ebay.com, imdb.com also support this schema to show a better result in search engine.

Schema.org defines several “vocabularies” for website owners to implement such as event, book, movie, health, etc. It also supports many different kind of format for user to implement such as Microdata, RDFa, JSON-LD.

2.11 Web Scraping challenges

While implementing web scraper is not a difficult simple task, there are several issues that need to be addressed.

Firstly, traditional web scrapers can only target certain XPath or CSS selector in HTML markup but websites can have new features added anytime which cause changes to the entire website. Thus, web scraper developers need to update their code with new correct selectors making the cost for maintaining and monitoring web scraper higher than building one. Guojun et al. (2017) look at a way to improve crawler for data extraction from dynamic web pages [18]. They have somewhat similar goals to Ujwal et al (2017) except that the focus in on getting relevant web pages rather than extracting specific elements from them [19]. In addition to that, they suggest techniques to better maintain and develop the crawler itself. Like Ujwal et al. (2017), Guojun et al. (2017)

attempt to solve the problem of mutating site structure [18]. The crawler has to intelligently choose which XPath pattern to regulate its data grabbing behavior. This is similar to the use of CSS-selector in Ujwal et al. (2017) paper.
Chaulagain et al. (2017) try to address the same problem covered in Guojun et al. and Ujwal et al. papers by building a smart scraper to get data from sources that have mutating document structure [19]. The use of XPath to extract relevant data is very similar to that of Guojun et al. (2017). The community schema support for Web Scraper also tries to solve this problem. However, while it is simple to understand how those schemas work, it is quite difficult to implement them into the website.

Secondly, as machine learning becoming ubiquitous, it is not unusual for websites to display personalized content. Web scraper cannot recognize which data is the correct data that it needs to retrieve. Many websites also check user IP Address to serve localized content. Web Scraper might extract content completely different from what normal user would see because their IP addresses are from different countries.

In summary, web scraper is a robust system that involves many technical components. The scraper needs to be able to understand markup language in the same way a browser does. It may also require that the scraper has its own browser engine in order to render the content correctly. Sometimes the wanted content can also be in other text and binary format so the scraper needs special components to extract data from there. The architecture of a web scraper is discussed. It is also important to plan on the data to collect and the pipeline needed to process it.

3. Web Scraping projects

To understand the process of building and deploying applications backed by Web Scraping in real life, two projects with different purposes and architectural approaches are implemented.

All projects are only used for this study and source code are available in Github public under MIT license. Projects are built, tested and deployed automatically by CircleCI. Python Programming Language with Scrapy framework is used to build the backend services and Web Scraper.

3.1 Electronics e-commerce search results comparison

3.1.1 Motivation

Product comparison is one of the most common types of application that usually uses Web Scraping. This project will try to scrape three popular online electronics stores in Finland: Gigantti, Verkkokauppa, and JIMMS. In this case, Web Scraping is built as a service and provides real-time scraped results to users through a simple web UI.

3.1.2 Application design

Based on the project requirements, the application backend need to be able to handle real time web scraping. The simplified application architecture is shown in Figure 5.

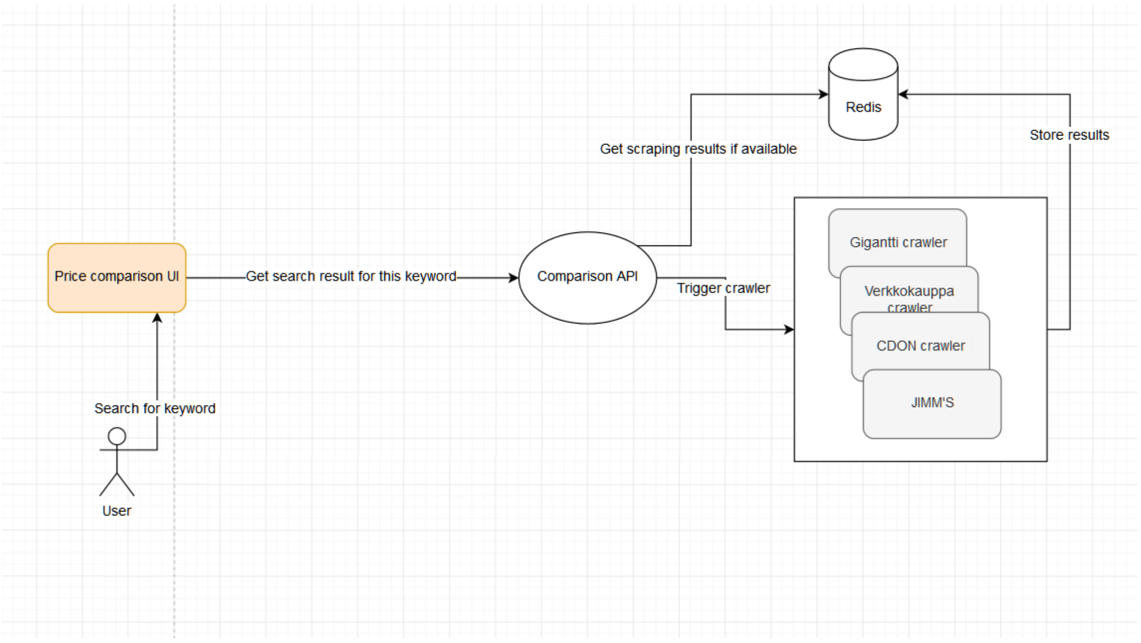


Figure 5. Diagram showing product comparison application architecture

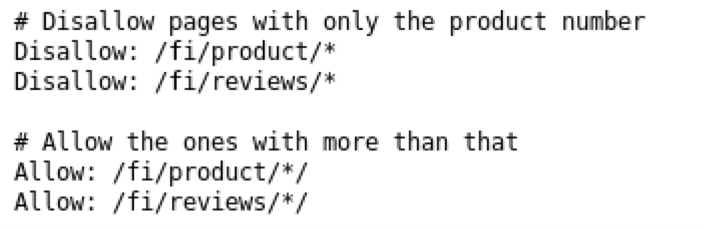
As seen in Figure 5, the application has a simple UI for users to enter search keywords. The UI sends search requests to Web Scraping service which then trigger Web scraper to scrape and return search results from Gigantti, Verkkokauppa, and JIMMS. This process might take more than one minute to finish which is a long time for users to wait. To mitigate this problem, a Redis database is used to cache scraping results. If two users request results for the same keyword, cached data in Redis will be used. In real life situation, a list of well-known keywords will be run against Web Scraping service every day to make sure that user experiences faster response time.

3.1.3 Scraping targets investigation and planning

To implement any Web Scraper, it is important to learn about the target website structure and robots.txt. All information listed below is fetched on the 1st of March 2019. The detail of robots.txt and website structure might change.

3.1.3.1 Verkkkokauppa

Verkkokauppa has a very long robots.txt but the most important part is shown in Figure 6.



```
# Disallow pages with only the product number
Disallow: /fi/product/*
Disallow: /fi/reviews/*

# Allow the ones with more than that
Allow: /fi/product/*/
Allow: /fi/reviews/*/
```

Figure 6. A part of robots.txt from Verkkokauppa

Based on information seen in Figure 6, scraping of detailed product page is not allowed because the product page path is always /fi/product/productname.html. However, this project only targets search result page (/fi/search?query=keyword) which is not specified in robots.txt.

During the time this paper is written, Verkkokauppa web structure changes two times. After recent changes, it is not easy to scrape more than 12 products from the search results without using a real Web browser with Selenium. Since this is only for research purposes, 12 first products should be enough.

For Verkkokauppa website, search results for 12 first products are returned in HTML response whenever user go to /fi/search?query=some-keyword. One trick to find out if the page content is just HTML returned by the server or JavaScript manipulation is to disable JavaScript with a web browser add-on. By going to the next page of the search result page, developers can find out that the result page can be controlled with parameter *page=number.* The results of disabling JavaScript can be seen in Figure 7.

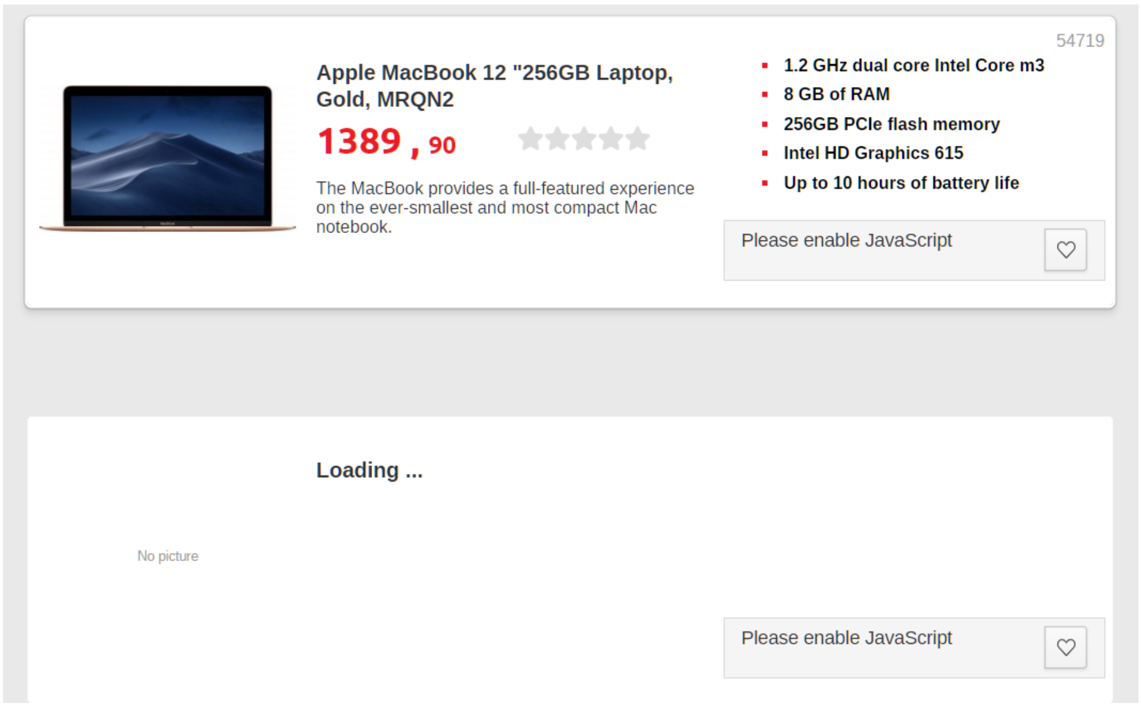
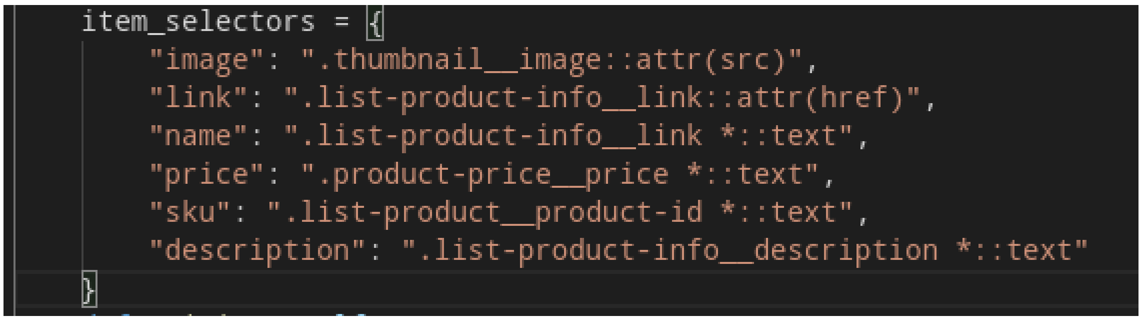

54719
Apple MacBook 12 "256GB Laptop, Gold, MRQN2
1389 , 90
The MacBook provides a full-featured experience on the ever-smallest and most compact Mac notebook.
1.2 GHz dual core Intel Core m3
8 GB of RAM
256GB PCIe flash memory
Intel HD Graphics 615
Up to 10 hours of battery life
Please enable JavaScript
Loading ...
No picture
Please enable JavaScript

Figure 7. Verkkokauppa search result page after JavaScript is disabled

After 12 results, only “Loading” text is there as shown in Figure 7. This also means that to get the first 12 results, the scraper only needs to send a request to that URL and parse HTML response in order to get product details.

Once the URL to request is figured out, it is crucial to know how to parse the content. Using the Web browser developer tool, developers can find the correct CSS or XPath selector. A list of CSS selector for typical product detail is shown in Figure 8.

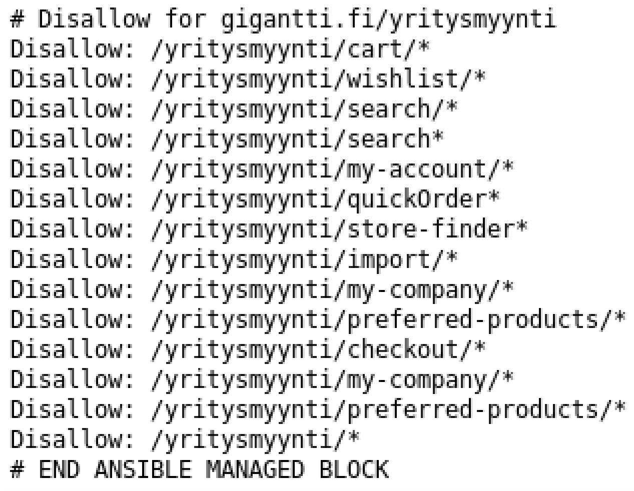


```
item_selectors = {
    "image": ".thumbnail__image::attr(src)",
    "link": ".list-product-info__link::attr(href)",
    "name": ".list-product-info__link *::text",
    "price": ".product-price__price *::text",
    "sku": ".list-product__product-id *::text",
    "description": ".list-product-info__description *::text"
}
```

Figure 8. CSS selector for Verkkokauppa product

3.1.3.2 Gigantti

A detail description of robots.txt is shown in Figure 9.



```
# Disallow for gigantti.fi/yritysmyynti
Disallow: /yritysmyynti/cart/*
Disallow: /yritysmyynti/wishlist/*
Disallow: /yritysmyynti/search/*
Disallow: /yritysmyynti/search*
Disallow: /yritysmyynti/my-account/*
Disallow: /yritysmyynti/quickOrder*
Disallow: /yritysmyynti/store-finder*
Disallow: /yritysmyynti/import/*
Disallow: /yritysmyynti/my-company/*
Disallow: /yritysmyynti/preferred-products/*
Disallow: /yritysmyynti/checkout/*
Disallow: /yritysmyynti/my-company/*
Disallow: /yritysmyynti/preferred-products/*
Disallow: /yritysmyynti/*
# END ANSIBLE MANAGED BLOCK
```

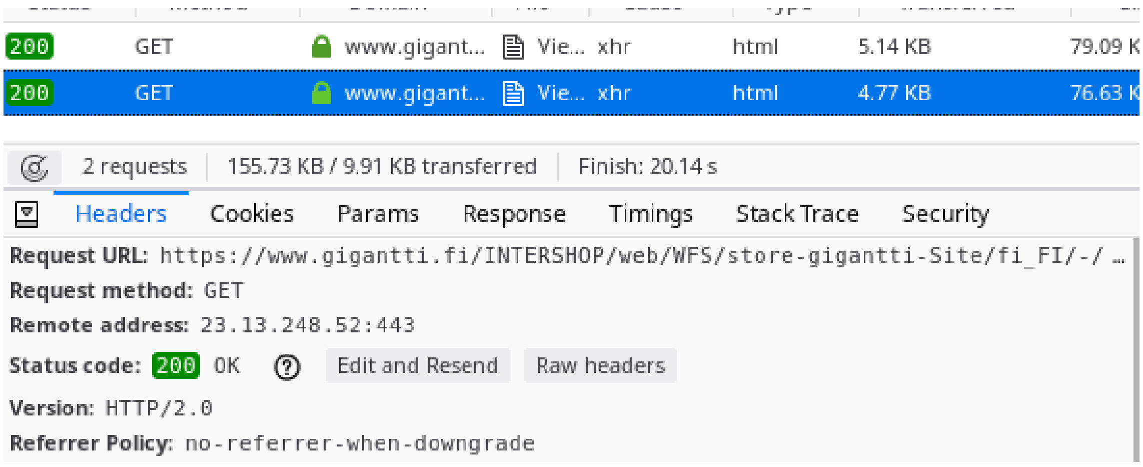
Figure 9. CSS selector for Verkkokauppa product

It can be seen from Figure 9 that Gigantti's robots.txt seems to disallow scraping search result at */yritysmyynti/search*

After doing some check on the website, it is clear that the search path that Gigantti uses is a different one at */search?SearchTerm=adidas* and it does not seems to be prohibited in robots.txt and therefore it is safe to proceed.

Interacting with the search result page shows that not all products data are returned once the user clicks 'search'. Only when the user scrolls down, more results are loaded. Similar to Verkkokauppa, disabling JavaScript show that only 6 products are returned as HTML when the browser makes a request to */search?SearchTerm=adidas.* To scrape more than 6 products, the developer needs to figure out how the page fetches more products when the user scrolls down.

In order to figure out what actually happens in the application when the user scrolls down, the developer needs to use the browser developer tool. Figure 10 shows the AJAX request logged in Chrome developer tool when user scrolls down for more search result.


```
200  GET  www.gigant...  Vie...  xhr  html  5.14 KB  79.09 K
200  GET  www.gigant...  Vie...  xhr  html  4.77 KB  76.63 K

2 requests   155.73 KB / 9.91 KB transferred   Finish: 20.14 s

Headers  Cookies  Params  Response  Timings  Stack Trace  Security

Request URL: https://www.gigantti.fi/INTERSHOP/web/WFS/store-gigantti-Site/fi_FI/-/ …
Request method: GET
Remote address: 23.13.248.52:443
Status code: 200 OK   Edit and Resend   Raw headers
Version: HTTP/2.0
Referrer Policy: no-referrer-when-downgrade
```

Figure 10. AJAX request sent to load more products when user scrolls down

As shown in Figure 10, an AJAX request is sent to */INTERNAL/web/WFS/store-gigantti-Site/fi_FI/EUR/ViewParametricSearchBySearchIndex-OfferPaging* when the page need to load more search results. Fortunately, the path in this AJAX request is not disallowed in robots.txt so it can be used to scrape more than 6 products from Gigantti. However, it also means that the logic that needs to be implemented for Gigantti scraper will be more complicated.

Even though */search?SearchTerm=adidas* and */INTERNAL/web/WFS/store-gigantti-Site/fi_FI/EUR/ViewParametricSearchBySearchIndex-OfferPaging* paths are different, both of them return HTML response and the product list HTML structure is similar. CSS selectors can be found easily. Figure 11 shows the list of CSS selector to extract product detail from Gigantti.

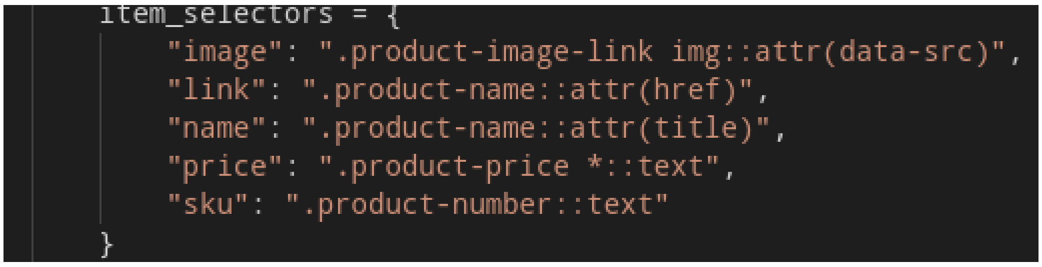
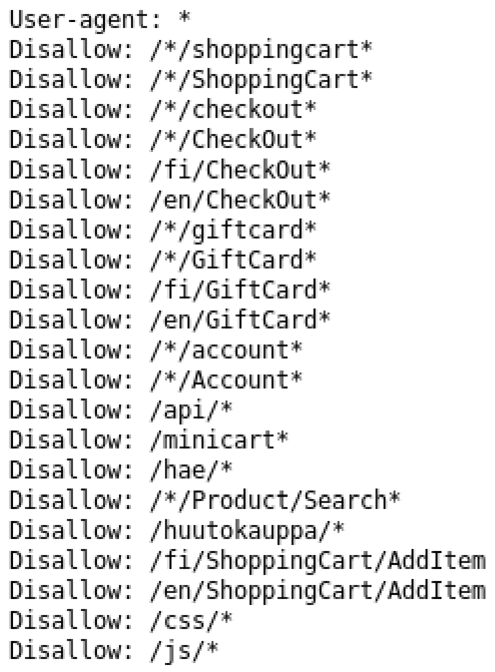

```
item_selectors = {
    "image": ".product-image-link img::attr(data-src)",
    "link": ".product-name::attr(href)",
    "name": ".product-name::attr(title)",
    "price": ".product-price *::text",
    "sku": ".product-number::text"
}
```

Figure 11. CSS selector for Verkkokauppa product

3.1.3.3 JIMMS

Figure 12 shows detail description of JIMMS robots.txt.



```
User-agent: *
Disallow: /*/shoppingcart*
Disallow: /*/ShoppingCart*
Disallow: /*/checkout*
Disallow: /*/CheckOut*
Disallow: /fi/CheckOut*
Disallow: /en/CheckOut*
Disallow: /*/giftcard*
Disallow: /*/GiftCard*
Disallow: /fi/GiftCard*
Disallow: /en/GiftCard*
Disallow: /*/account*
Disallow: /*/Account*
Disallow: /api/*
Disallow: /minicart*
Disallow: /hae/*
Disallow: /*/Product/Search*
Disallow: /huutokauppa/*
Disallow: /fi/ShoppingCart/AddItem
Disallow: /en/ShoppingCart/AddItem
Disallow: /css/*
Disallow: /js/*
```

Figure 12. JIMMS robots.txt file showing prohibition for almost all path

From Figure 12, it can be seen that JIMMS prohibits any scraping activity. Further research into the website shows that the products search URL is /fi/Product/Search and that path scraping is indeed restricted. However, the website is a good place to learn the way to scrape a web application completely rendered by JavaScript. When JavaScript is disabled, the website stop working completely unlike Verkkokauppa and Gigantti.

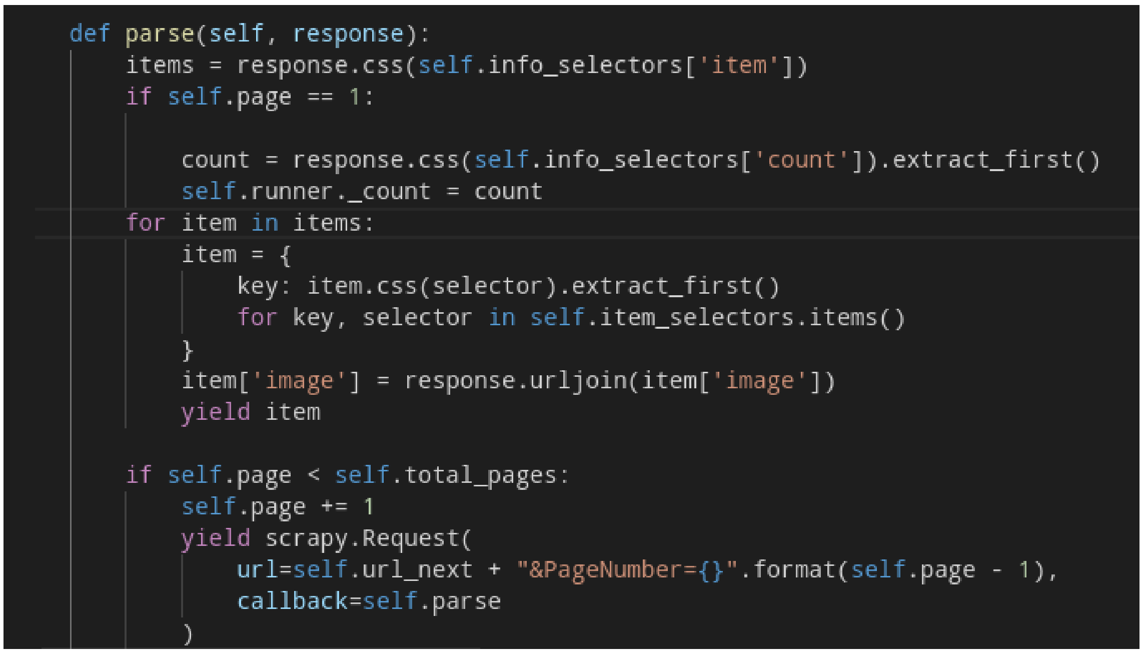
Using Web Browser Developer Tool, it is straightforward to discover that when users trigger the search button, a request is sent to https://www.jimms.fi/api/product/searchloop to retrieve the list of products in JSON format. The web application then renders the page with that data. Even though the web application is quite complicated, it is much easier to scrape JIMMS through its API bypassing the application. Due to the fact that search page scraping is forbidden in robots.txt, JIMMS scraper won't be implemented for this project. As discussed in Web scraping rulebook section, it would be better to be nice and follow what website owner defined in robots.txt.

3.1.4 Implementation

After the investigation, it was decided that only Verkkokauppa and Gigantti scrapers will be implemented due to robots.txt.

One important thing is that robots.txt might change anytime so the implemented scrapers all need to check robots.txt before doing any scraping to be compliance. Fortunately, Scrapy framework can be configured to comply with robots.txt automatically. Furthermore, the framework also allows setting a random interval between each request to target website to avoid being noisy.

Building Web Scraper with Scrapy is quite simple. The most important is to tell the scraper how to parse the response and how to request the next page. Figure 13 show an example parsing function for Gigantti.



```
def parse(self, response):
    items = response.css(self.info_selectors['item'])
    if self.page == 1:

        count = response.css(self.info_selectors['count']).extract_first()
        self.runner._count = count
    for item in items:
        item = {
            key: item.css(selector).extract_first()
            for key, selector in self.item_selectors.items()
        }
        item['image'] = response.urljoin(item['image'])
        yield item

    if self.page < self.total_pages:
        self.page += 1
        yield scrapy.Request(
            url=self.url_next + "&PageNumber={}".format(self.page - 1),
            callback=self.parse
        )
```

Figure 13. Parser function for Gigantti scraper

As described in the application's design section, the Web Scraper need to be trigger by an API call. Because Web Scraping asynchronous nature, the API also needs to return results asynchronously or a complex task queue system needs to be implemented. The Twisted Klein Web API framework supports asynchronous response by default and is a perfect combination for Scrapy Web Framework.

The entire Web API and scrapers are then packaged inside a Docker image. This also enables easy deployment of multiple instances of the API and scrapers. With the power of Docker, the application is easy to scale and is more fault tolerant.

3.2 Simple search engine application

3.2.1 Motivation

A search engine, especially Google, is an essential part of the Internet. While it is fine to take it for granted and use what is available, it is quite interesting to create your personal search engine. In this project, a simple search engine will be created with the help of Web Scraping. Most data used will be scraped from reddit.com, 4chan.org, medium.com, quora.com and stackoverflow.com and any websites linking to them and stored in ElasticSearch database. ElasticSearch is a database engine that supports full text search on text documents. Since the scope of this paper is not about search algorithms, the default search algorithm that ElasticSearch support will be used.

3.2.2 Application design

The web UI and API parts are basically the same as the product comparison project. In fact, in the code stored in Github, they share the same component. The part that should be the focus is the Web Scraper. Figure 14 shows the architecture design of the application.

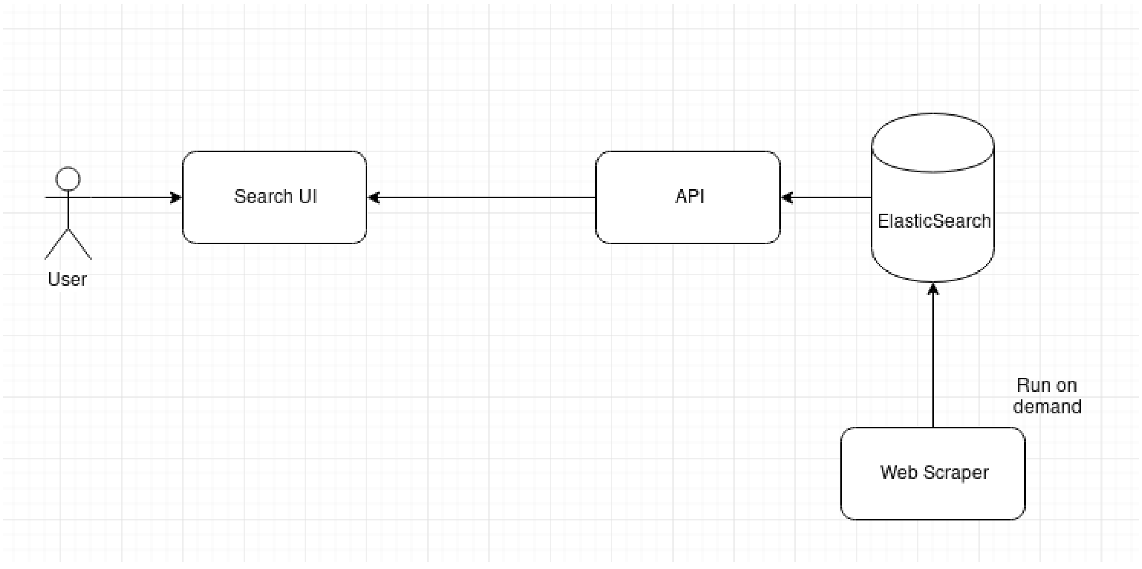


Figure 14. Simple search engine (DSearch) architecture diagram

As shown in Figure 14 and Figure 5, the only difference between this application and the first one is ElasticSearch and Web Scraper. Implementation detail will be discussed in the next section.

3.2.3 Implementation

Unlike the first comparison project where Web Scraper is implemented as a Service and return results in real time, the Web Scraper, in this case, is just a CronJob or a Job that will be triggered to scrape data and push them to ElasticSearch. One major difference is that this Web Scraper does not know detail about how to parse specific website but instead try to extract as much useful text as possible. This is where schema.org common HTML markdown is useful. However, both Reddit and 4chan do not follow schema.org convention. To extract as much text as possible from any web pages, it is decided to go with a naive approach that the Web Scraper will extract text from any <p> HTML tag and set it as content to be pushed to ElasticSearch. All links in <a> HTML tag inside web pages are also extracted and followed.
Figure 15 shows how web scraper discovers a new website and follows to the next one.

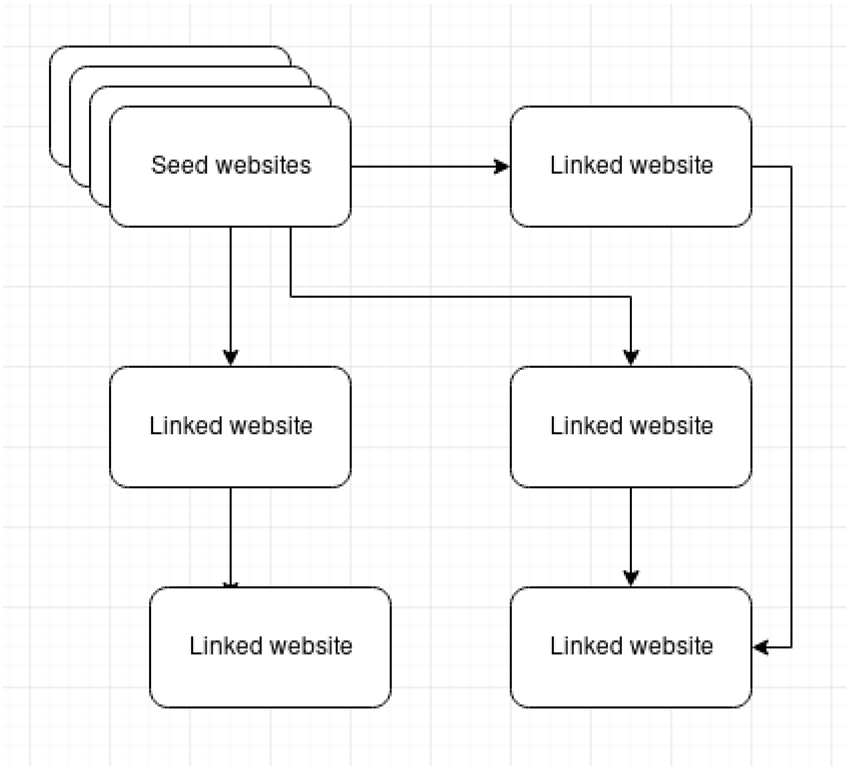


Figure 15. Web Scraper starts with seed websites and goes to all linked websites

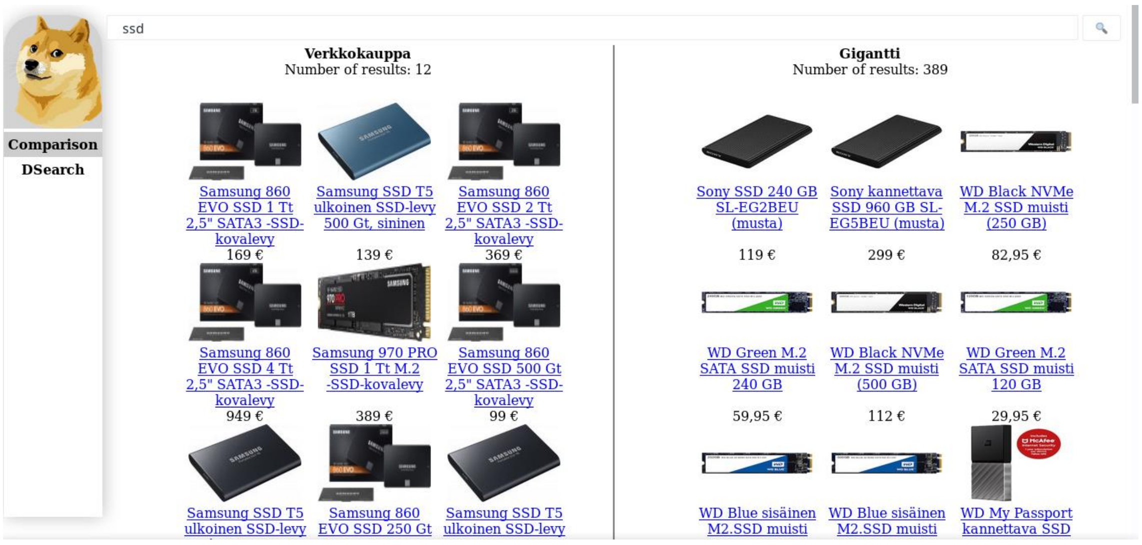
As shown in figure 15, the first few websites that the Web Scraper visits are called seed websites, they are websites with many external links that Web Scraper can detect and use to connect to more websites. However, the internet is very big there can be millions of links. To avoid scraping too many websites, it is advised to limit the following

algorithm to two or three consecutive linked websites. Scrapy Scraping Framework supports the setting out of the box.
Since this Web Scraper does not know details about how to scrape a specific website, the extracted data could be polluted with JavaScript, CSS or even some malicious code. It is important to do proper checks to clean and remove codes from text data. For simple cases, a Python library called *w3lib.html* can be used.

4. Results discussion

To demonstrate the results of Web Scraping from implemented projects, a simple web UI is created. In the UI the user can choose which application to try out.

The final user interface for product comparison project is shown in Figure 16. Detail search results from Verkkokauppa and Gigantti are scraped and display based on user input.



Verkkokauppa
Number of results: 12

Samsung 860 EVO SSD 1 Tt 2,5" SATA3 -SSD-kovalevy
169 €

Samsung SSD T5 ulkoinen SSD-levy 500 Gt, sininen
139 €

Samsung 860 EVO SSD 2 Tt 2,5" SATA3 -SSD-kovalevy
369 €

Samsung 860 EVO SSD 4 Tt 2,5" SATA3 -SSD-kovalevy
949 €

Samsung 970 PRO SSD 1 Tt M.2 -SSD-kovalevy
389 €

Samsung 860 EVO SSD 500 Gt 2,5" SATA3 -SSD-kovalevy
99 €

Samsung SSD T5 ulkoinen SSD-levy

Samsung 860 EVO SSD 250 Gt

Samsung SSD T5 ulkoinen SSD-levy

Gigantti
Number of results: 389

Sony SSD 240 GB SL-EG2BEU (musta)
119 €

Sony kannettava SSD 960 GB SL-EG5BEU (musta)
299 €

WD Black NVMe M.2 SSD muisti (250 GB)
82,95 €

WD Green M.2 SATA SSD muisti 240 GB
59,95 €

WD Black NVMe M.2 SSD muisti (500 GB)
112 €

WD Green M.2 SATA SSD muisti 120 GB
29,95 €

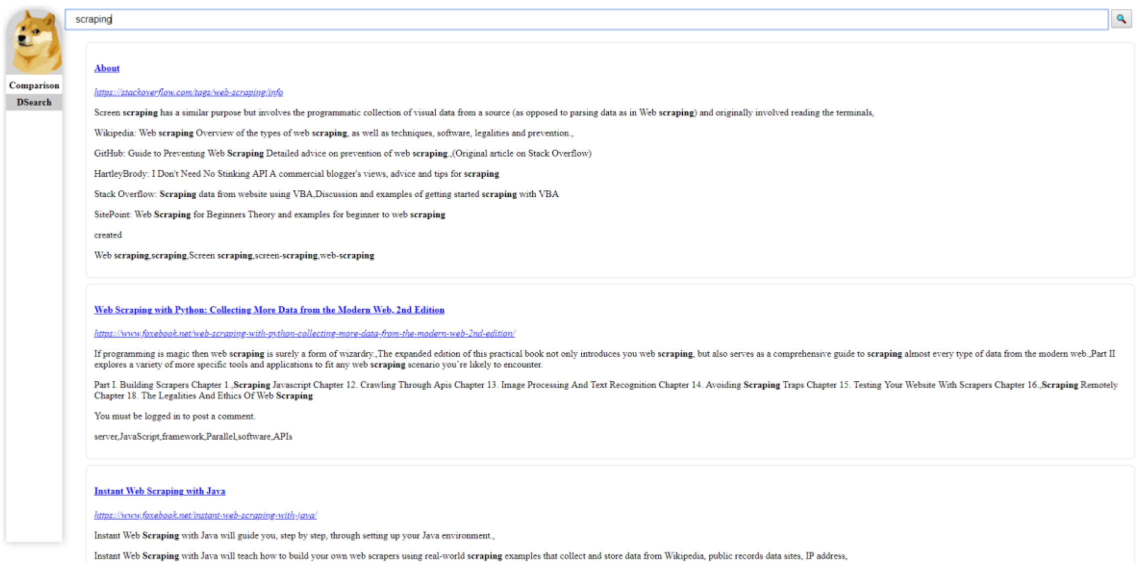
WD Blue sisäinen M2.SSD muisti

WD Blue sisäinen M2.SSD muisti

WD My Passport kannettava SSD

Figure 16. Product comparison interface

Figure 17 shows full text search interface of the second application. The application interface is similar to Google search interface. A list of search result containing website title, origin and text matching result is shown based on user input.



About

https://stackoverflow.com/tags/web-scraping/info

Screen **scraping** has a similar purpose but involves the programmatic collection of visual data from a source (as opposed to parsing data as in Web **scraping**) and originally involved reading the terminals,

Wikipedia: Web **scraping** Overview of the types of web **scraping**, as well as techniques, software, legalities and prevention.,

GitHub: Guide to Preventing Web **Scraping** Detailed advice on prevention of web **scraping**.,(Original article on Stack Overflow)

HartleyBrody: I Don't Need No Stinking API A commercial blogger's views, advice and tips for **scraping**

Stack Overflow: **Scraping** data from website using VBA,Discussion and examples of getting started **scraping** with VBA

SitePoint: Web **Scraping** for Beginners Theory and examples for beginner to web **scraping**

created

Web **scraping**,**scraping**,Screen **scraping**,screen-**scraping**,web-**scraping**

Web Scraping with Python: Collecting More Data from the Modern Web, 2nd Edition

https://www.foxebook.net/web-scraping-with-python-collecting-more-data-from-the-modern-web-2nd-edition/

If programming is magic then web **scraping** is surely a form of wizardry.,The expanded edition of this practical book not only introduces you web **scraping**, but also serves as a comprehensive guide to **scraping** almost every type of data from the modern web.,Part II explores a variety of more specific tools and applications to fit any web **scraping** scenario you're likely to encounter.

Part I. Building Scrapers Chapter 1.,**Scraping** Javascript Chapter 12. Crawling Through Apis Chapter 13. Image Processing And Text Recognition Chapter 14. Avoiding **Scraping** Traps Chapter 15. Testing Your Website With Scrapers Chapter 16.,**Scraping** Remotely Chapter 18. The Legalities And Ethics Of Web **Scraping**

You must be logged in to post a comment.

server,JavaScript,framework,Parallel,software,APIs

Instant Web Scraping with Java

https://www.foxebook.net/instant-web-scraping-with-java/

Instant Web **Scraping** with Java will guide you, step by step, through setting up your Java environment.,

Instant Web **Scraping** with Java will teach how to build your own web scrapers using real-world **scraping** examples that collect and store data from Wikipedia, public records data sites, IP address,

Figure 17. The full-text search engine interface

For the first application which shows a real-time comparison from Gigantti and Verkkokauppa, implementation of scrapers has been effortless. However, during the time this paper is written, several changes happen to target scraping websites including website structure and robots.txt. The changes to Verkkokauppa cause the current web scraper to only return the first 12 products. Robots.txt changes in JIMMS website prevent scrapers from accessing it entirely. It is challenging to guarantee good user experiences because the application has to show real time results. The application tends to break if it is not maintained regularly. Due to the requirements to scrape different types of data product price, images, description, title, Web Scraper need to know specifically where and how to scrape such data. Any changes in the scraping target can cause errors in the Scraper.

The second application, on the other hand, requires complicated parsing logic and data cleaning. Parsed data currently is limited to plain text. Once the data is cleaned up and pushed to the database, it is difficult to recognize what the text is about and extract small details such as price from the text. Furthermore, the application tries to store as much text as possible so it requires much more storage space. Scraped data tends to be out of date because data is scraped from days or weeks before and not real time. The application also requires much more resources to be production ready. One interesting result from this application Web Scraping is that even though the Scraper only run for a few days with 5 seed websites: stackoverflow.com, medium.com, 4chan.org, quora.com, reddit.com, there are a lot of scraped data on other websites that somehow linked to those 5 websites such as bcc.com, w3school. However, after a week of scraping, the results became repetitive and not much new data was saved. This might be because the list of seed websites is too small. The project also has limited resource running Scraping on a laptop. It is more ideal in the real world to distribute this kind of scraping to multiple machines.

In the end, the process of building a reliable web scraper is proven to be more difficult than is initially thought. The maintainer of the scraper system should always be aware of the changes made to the target site and change the retrieval logic accordingly. Some site owners are also against web scraping and thus made clear in the policy that no robot scraper could visit their site in part or completely. Since the volume of data is sometimes big, it is important to have reasonable rate limit and other smart mechanisms to optimize the payload over the network. The analysis step of the application is also complicated. It could be improved by using some AI language recognition to pick out the relevant information from the raw data. But that is not in the scope of this paper.

5. Conclusion

With the growth of internet usage, the amount of data available online is exploding. To efficiently extract meaningful data from the vast ocean of web contents, intelligent systems called scrapers were built to automate the process. Knowing how a scraper works and how to make an efficient scraper can help not just business but also individuals who want to get just the right information. As the economy moves towards hi-tech industry, it is more important to retrieve the necessary data in a reliable and timely manner. Web scraper has great potential to grow as a tool for navigating the abundant and unstructured sources of data.

As expected from the theoretical background on web scrapers, there are many challenges in the process of developing and maintaining a web scraper. The implementation in this paper indeed had to use a browser engine to be able to reach certain parts of the target site. Fortunately, there was no need to use any technique for data extraction from image or binaries. The paper also discusses a simple pipeline for data analysis, though in practice it is likely to be a lot more challenging. It is also demonstrated that some website owners are aware and have concerns about web scrapers looking into their sites. Thus, it is important to make a scraper that complies with the robots policy of the site.

Web Scraping is a powerful technology that is widely used to automate the data gathering process around the Internet and powers web applications with data. While implementing Web Scraping is not a difficult task, there are challenges in using one. The cost to maintain such Web Scraper systems could be more than buying the data from a reliable data source. Furthermore, developers can run into legal issues if they are not careful. This study showed that it is important to write nice and polite Web Scraper that follows the rules and even ask the owner for permissions if necessary.

References

1. https://en.wikipedia.org/wiki/HTML. Accessed on March 2019
2. https://en.wikipedia.org/wiki/Cascading_Style_Sheets. Accessed on March 2019
3. https://en.wikipedia.org/wiki/JavaScript. Accessed on March 2019
4. https://developer.mozilla.org/en-US/docs/Web/XPath. Accessed on March 2019
5. https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Selectors. Accessed on March 2019
6. http://elementalselenium.com/tips/34-xpath-vs-css-revisited-2. Accessed on March 2019
7. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model. Accessed on March 2019
8. https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction. Accessed on March 2019
9. Mahto D, Singh L. A Dive into Web Scraper World. Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on March 2016 : 689-693
10. Upadhyay S, Pant V, Bhasin S, Pattanshetti M. Articulating the Construction of a Web Scraper for Massive Data Extraction. 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT) in February 2017
11. Lawson R. Web Scraping With Python. October 2015
12. Haque A, Singh S. Anti-scraping application development. Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on August 2015
13. Scrapy documentation. https://doc.scrapy.org/en/latest/ .Accessed on March 2019
14. Heydt M. Python Web Scraping Cookbook. Feb 2018
15. Narayanan A, Shmatikov V. Robust De-anonymization of Large Sparse Datasets. University of Texas on 2018.

16. Chaulagain RS, Pandey S, Basnet SR. Cloud Based Web Scraping for Big Data Applications. Smart Cloud (SmartCloud), 2017 IEEE International Conference on November 2017
17. https://schema.org/. Accessed on March 2019
18. Guojun Z, Wenchao J, Jihui S. Design and application of intelligent dynamic crawler for web data mining. Automation (YAC), 2017 32nd Youth Academic Annual Conference of Chinese Association on May 2017
19. Ujwal B V S, Gaind B, Kundu A. Classification-Based Adaptive Web Scraper. Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on December 2017