Towards Data Extraction of Dynamic Content from JavaScript Web Applications

Korawit Prutsachainimmit
College of Computing
Prince of Songkla University
Phuket, Thailand
korawit.p@phuket.psu.ac.th

Winai Nadee
College of Computing
Prince of Songkla University
Phuket, Thailand
winai.na@phuket.psu.ac.th

***Abstract*— An enormous data in World Wide Web and social media has open opportunities for business and organization to get the significant value that leads to efficient operations. As a result, Web Data Extraction has become an important tool for gathering and translating semi-structured documents into valuable information. However, one of the major challenges is dealing with changes from Web documents, especially emerging of JavaScript Web development technology that has significantly affected the way to embed and rendering data of Web pages. In this paper, we propose a design and implementation of a new Web Data Extraction system that aims for extracting data from JavaScript Web applications. The proposed system enables users to select valuable data from online Web documents by defining data extraction rules and data transformation patterns. The extraction engine automatically scrapes and transforms semi-structure data into relational data. The preliminary evaluation results showed that our proposed system has successfully extract data from modern JavaScript Web applications.**

Keywords— Information retrieval; Web Data Extraction; JavaScript Web application; JSON;

I. INTRODUCTION

With the explosive growth of the World Wide Web and social media, a tremendous of data has become available online in the form of text, photos, videos, etc. This situation opens the opportunity for users to benefit from the available information in many interesting ways, especially data analytics. With analytics of big data from World Wide Web and social media, business and organization can offer significant value that leads to efficient operations, higher profits and better customer's experience. However, the information on the World Wide Web and social media is mainly designed for human browsing, not in a structured form that can be used by other applications. Though many data sources are publicly available as Web services, many others data sources are still not accessible through a programming interface. As a result, the technologies for extracting data from Web documents, so-called Web data extraction, have become an important tool for gathering data [1].

Web Data Extraction systems are software applications aiming at extracting data from Web documents and translate them into structured data. The common process involves fetching and extracting. Fetching is a process of downloading a Web page, which is similar to what browser does when a user browses a Web page. Extracting is done later when the target documents are downloaded. The downloaded documents may be searched, parsed, formatted and then transformed into structured data. The process of fetching and extracting is usually done automatically and repeatedly in order to deliver extracted data into a spreadsheet, database or some other application [2].

One of the major challenges in Web Data Extraction is dealing with changes of Web document over time. Evolving of Web development technologies is a key factor that affects the structure of Web documents. For instance, emerging of JavaScript frameworks in Web development has significantly changed the way of embedding data and the rendering process of Web pages.

In cases of extracting data from a traditional Web application, the extraction process begins with sending a request to the target server. In every request, the server renders data, embed and formatted in an HTML document, and responses it back to the client. With this method, a Web Data Extraction tool can normally process and extract data from the downloaded documents. In contrast to the traditional Web applications, modern JavaScript Web applications fetch data from Web servers through asynchronous JavaScript calls. The server returns only data (no HTML markup) to the client in an asynchronous manner. JavaScript code in client browsers uses the received data to construct the page dynamically. As a result, the client browser or Web Data Extraction process cannot receive the HTML documents and accesses DOM of the target Web pages with the ordinary method. Thus, extracting data from modern JavaScript Web applications is a new challenge for designing and developing a Web Data Extraction tool.

In this paper, we propose a semi-automatic Web Data Extraction system that aims for extracting data from modern JavaScript Web applications. The proposed system is designed to enable end-users to select data from existing Web documents by defining data extraction rules and data transformation patterns. The extraction process is invoked automatically following the user-defined schedule. We have tested our proposed system by extracting product information from an online shopping Website.

II. BACKGROUND AND RELATED WORK

A. *Web Data Extraction*

Web Data Extraction system has been used in a variety of applications including document analysis, business intelligence, social media, analytics, etc. Systems and tools are developed for extracting data from unstructured documents (emails, business forms or technical papers) and semi-structured documents i.e. Web documents [3]. In this research, we focus on extracting data from Web documents which are massive of semi-structured information presented in HTML formats. To efficiently pull out data from HTML documents, the existing system adopts the broad class of techniques that is text processing, DOM parsing, natural language processing and machine learning [4, 5]. The high-efficiency algorithms and state of the art approaches are implemented as open-source libraries and commercial software.

The designing of Web Data Extraction consists of two parts that apply two different techniques. The first part is algorithms for extracting data from HTML documents. Since a Web document is a hierarchy of HTML elements that are usually represented as DOM (Document Object Model), most of data extraction systems rely on tree-based techniques i.e. addressing, matching and weighing of tree nodes [6]. The second part is called Wrapper, which is a procedure that implements one or multiple data extraction algorithms [7]. To carry on with data extraction process, wrapper continuously runs the data extraction algorithms, transforms and merges them into a structured format. Obviously, exploiting DOM of Web documents is the key mechanism in existing Web Data Extraction approaches.

B. *JavaScript Web Application*

The growing popularity of JavaScript has changed the way of developing Web applications. JavaScript Web applications evolve the process of rendering Web page and DOM manipulation to offer better Web browsing experience. One of the widely adopted architecture for JavaScript Web application is Single Page Application (SPA). SPA is web application that fully loads all resources in the initial request. The individual components, including DOM, can be replaced or updated independently depending on user's interaction [8]. The other technique that adds the interactive capability to SPA is Asynchronous JavaScript and XML (AJAX). Instead of making a new request and update the whole page with new data every time. SPA acquires only data, usually in JSON format, from the server by creating a background process for sending asynchronous requests to the server. When the requested data has arrived, SPA injects the data into HTML elements by dynamically manipulating DOM.

Since asynchronous data transfer and dynamic DOM manipulation is the key features of SPA and modern JavaScript Web applications, extracting data from this kind of Web applications has become a new limitation of existing Web Data Extraction system, which mainly relies on DOM processing. Therefore, this research aims to find an optimal solution which enables data extraction from modern JavaScript Web applications.

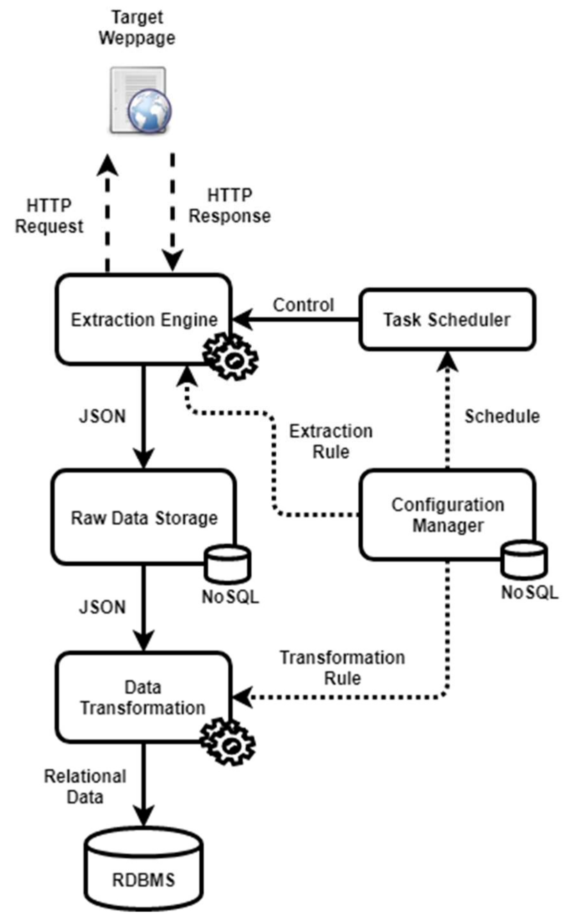


Fig. 1. An overview of the proposed architecture

III. PROPOSED APPROACH

The key idea of our approach is the new design and implementation of data extraction engine. We propose a new Web Data Extraction engine that utilizes the headless browser for fetching Web documents and dealing with dynamically generated DOM.

Rather than performing tree-based techniques on DOM, our approach focuses on extracting JSON data that is asynchronously transferred and cached inside the target Web documents. To accommodate end-users in the data extraction tasks, our approach implements a task scheduler for semi-automatic repeating the data extraction process. A configuration system is used to allow users to define extraction rules and other configurations. Data transformation is designed to translate semi-structured data into structured data that can be an input of data analytics process. In the following subsections, we describe the overview architecture of our approach. We also highlight the key ideas and techniques that we have applied to deal with challenges in enabling data extraction of JavaScript Web application.

A. *Overview of Architecture*

An overview of the proposed architecture shows in Fig 1. The proposed system consists of 5 major modules as the following:

Extraction Engine. Once the extraction process starts, Extraction Engine is responsible for fetching and extracting data corresponding to the extraction rules provided by Configuration Manager.

Task Scheduler. To control schedule and frequency of running Extraction Engine, Task Scheduler reads configuration from Configuration Manager and invokes Extraction Engine following the user-defined schedules.

Raw Data Storage. This module is a NoSQL data storage designed to keep the original JSON data which is the output of Extraction Engine. Once JSON data is extracted from a target Web document, Extraction Engine stores original version of JSON data in this storage.

Data Transformation. With pre-defined rules from Configuration Manager, JSON data from Raw Data Storage is transformed into relational data and stored in a relational database management system by this module. Since the transformation rules can be added or changed later, keeping raw data in a separated storage gives flexibility and benefits to data transformation process and future data analytics.

Configuration Manager. To enable semi-automatic data extraction and transformation, Configuration Manager provides user-defined configurations which include data extraction rules, invocation schedule and data transformation rules for supporting execution of other modules.

B. *Extraction Engine*

To deal with dynamically generated DOM used in modern JavaScript Websites, the extraction engine employs headless browser technique for fetching the target Web documents. In this way, other processes of Extraction Engine can extract data from DOM of the fetched documents as ordinary HTML documents that contain a static DOM. The detail process of Extraction Engine is illustrated in Fig 2.

During our experiment, by analyzing DOM of tested Websites, we found that modern JavaScript Websites acquire data from back-end Web services using JSON format and store the received JSON data in some part of its DOM as cached data. As a result, our approach leverages the embed JSON data for the data extraction process by using the DOM parser to extract the embed JSON data instead of getting text value inside HTML tags as implemented in the existing approaches. By extracting cached JSON data from the target Web documents, our proposed method can overcome the common problem of Web extraction engine, i.e. dealing with changing of DOM elements, because JSON data are rarely changed. Thus, the major task of Extraction Engine is finding and extracting JSON chunk of data that represents required information from the target Web pages.

The major challenge of Extraction Engine is finding cached JSON data in DOM of the documents. Our solution is allowing users to specify target data in the form of data extraction rules. An example of data extraction process is shown in Fig. 3. The processing flow of the example can be described as following:

Step 1. Read the DOM extraction rules from Configuration Manager.

Step 2. Parse the fetched documents to create DOM tree

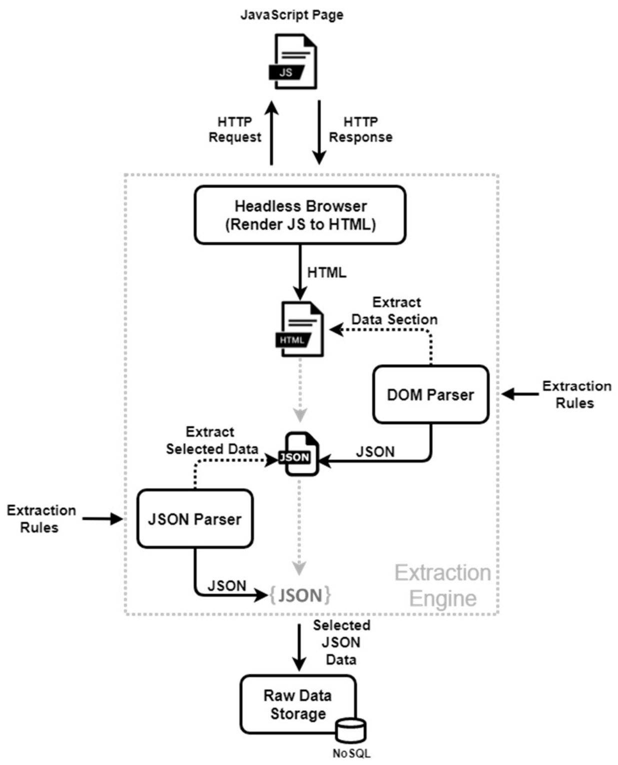
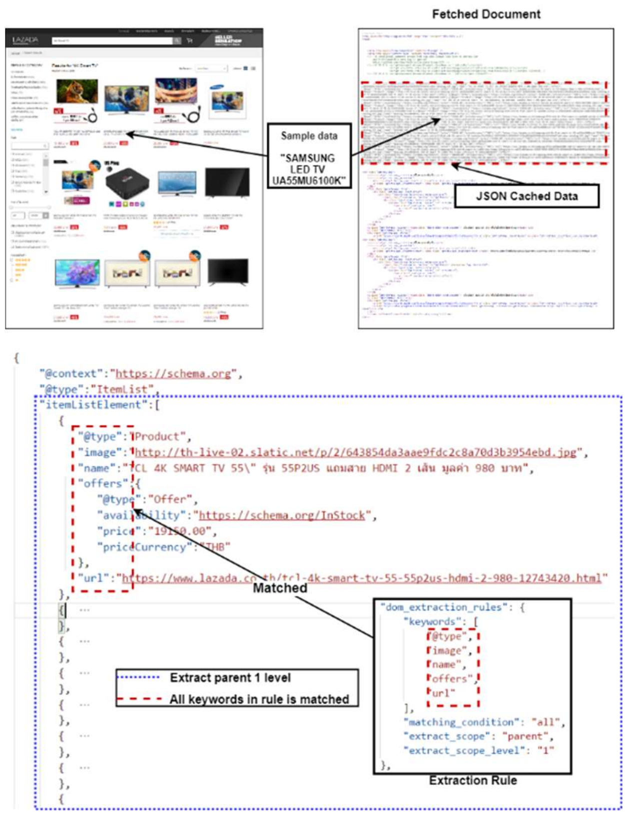


Fig. 2. The detail process of Extraction Engine



```
{
    "@context":"https://schema.org",
    "@type":"ItemList",
    "itemListElement":[
        {
            "@type":"Product",
            "image":"http://th-live-02.slatic.net/p/2/643854da3aae9fdc2c8a70d3b3954ebd.jpg",
            "name":"TCL 4K SMART TV 55\" รุ่น 55P2US แถมสาย HDMI 2 เส้น มูลค่า 980 บาท",
            "offers":{
                "@type":"Offer",
                "availability":"https://schema.org/InStock",
                "price":"19150.00",
                "priceCurrency":"THB"
            },
            "url":"https://www.lazada.co.th/tcl-4k-smart-tv-55-55p2us-hdmi-2-980-12743420.html"
        },
        {   ...
        },
        {   ...
        },
        {   ...
        },
        {   ...
        },
        {   ...
        },
        {   ...
        },
        {
```

```
"dom_extraction_rules": {
    "keywords": [
        "@type",
        "image",
        "name",
        "offers",
        "url"
    ],
    "matching_condition": "all",
    "extract_scope": "parent",
    "extract_scope_level": "1"
},
```

Fig. 3. An example of data extraction process

Step 3. Traverse each element in DOM tree.

Step 4. Compare the current DOM element with the value specified in the extraction rules.

According to the "matching_condition: all", DOM Parser traverses each element of the fetched documents and finds the element that contains all the values specified in the extraction rules i.e. "@type", "image", "name", "offers", and "url". If all the specific elements are found in a specific DOM element, DOM Parser extracts one parent level of JSON data from the target document, i.e. "itemListElements", which is the JSON collection that contains data of all products of the target document. Otherwise, move to the next DOM elements (Step 3).

Since the output JSON chuck may contain unrelated or uninterested data, the JSON parser is responsible for selecting data according to the criteria specified in the extraction rules. Finally, the selected JSON data is stored in the Raw Data Storage for the data transformation process.

C. *Configuration File*

The configuration file is defined as a well-formed JSON document that contains setting parameters for each module in a separated configuration section. The template of the configuration file and short explanation of configuration values in each section is illustrated in TABLE. I.

TABLE. I. TEMPLATE OF THE CONFIGURATION FILE

Configuration Sections

Target Web Document

```
"target": [
    {
        "base_url": "[URL of target Web document]",
        "parameters": [
            "[Parameter for completing URL of target Web document]"
        ],
        "params_matching": "[Parameter matching pattern (one-many / one-one)]"
    }
],
```

Task Scheduler

```
"schedule": {
    "interval": "[Frequency of extraction]",
    "unit": "[Unit of frequency]",
    "start_time": "[Time to start the extraction process]",
    "end_time": "[Time to stop the extraction process]",
    "no_limit": "[Stop the extraction process when this value is reached]"
},
```

DOM Parser

```
 "dom_extraction_rules": {
     "keywords": [
         "[Sample data for searching JSON chunk]"
     ],
     "matching_condition": "[Matching condition for keywords (all / one)]",
     "extract_scope": "[Scope of JSON chunk extraction (parent / top)]",
     "extract_scope_level": "[No of extraction level]"
 },
```

Data Transformation

```
"transfomation_rules": {
    "db_connection_name": "[RDBMS connection name]",
    "table_name": "[RDBMS table name]",
    "source": [
        "[JSON data fields]"
    ],
    "destination": [
        "[RDBMS columns]"
    ]
}
```

IV. IMPLEMENTATION

A. *System Development*

In order to enable users to easily extract data from Web documents and to experiment our solutions with the real-world JavaScript Web applications, we have developed our proposed system as a command-line application using Node.js. We have selected PhantomJS [9] for implementing the headless browser in Extraction Engine. MongoDB and MySQL are applied for NoSQL data storage and RDBMS respectively. A user can start the extraction process by calling the system via a command-line tool with a valid configuration file as a parameter. The output of the system is the relational data that is stored in a relational database corresponding to the configuration.

B. *Experimental*

We have conducted a preliminary evaluation of our system using a data extraction scenarios.

Scenario. *Extract product information from a shopping Websites:* This scenario simulates a data gathering task that aims to collect product information, including name, description, and prices, from a shopping Website. We selected Lazada [10], the top shopping online Website of Thailand, as the target Website for this scenario. The extraction rules and configurations were set to retrieve name, description and price of selected product categories. The expected extraction result is a time series of data that can express pricing trend and support price prediction in the future. The frequency of extraction was set to one time per day due to the discount campaign of Lazada which not commonly change within a day. An example of target Web document of this scenario shows in Fig. 4.

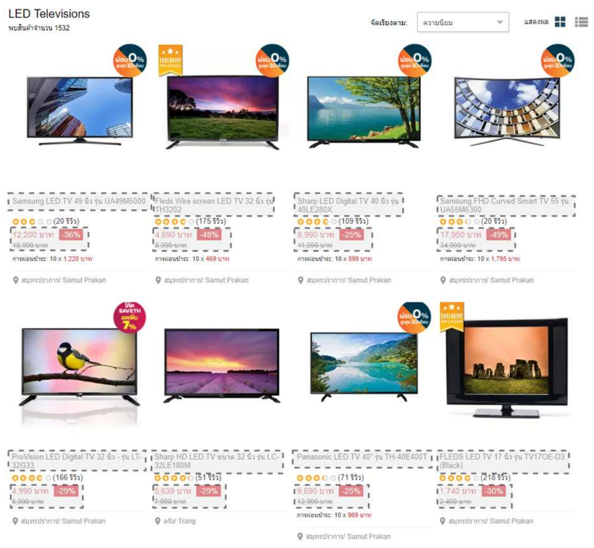
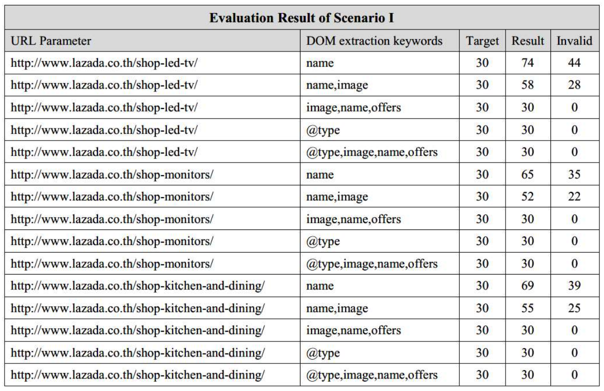

LED Televisions
Samsung LED TV 49 นิ้ว รุ่น UA49M5000
(20 รีวิว)
12,200 บาท -36%
18,900 บาท
การผ่อนชำระ: 10 x 1,220 บาท
สมุทรปราการ/ Samut Prakan
Fleds Wire screen LED TV 32 นิ้ว รุ่น TH3202
(175 รีวิว)
4,690 บาท -48%
8,990 บาท
การผ่อนชำระ: 10 x 469 บาท
สมุทรปราการ/ Samut Prakan
Sharp LED Digital TV 40 นิ้ว รุ่น 40LE280X
(109 รีวิว)
8,990 บาท -25%
11,990 บาท
การผ่อนชำระ: 10 x 899 บาท
สมุทรปราการ/ Samut Prakan
Samsung FHD Curved Smart TV 55 รุ่น UA55M6300
(20 รีวิว)
17,950 บาท -49%
34,990 บาท
การผ่อนชำระ: 10 x 1,795 บาท
สมุทรปราการ/ Samut Prakan
ProVision LED Digital TV 32 นิ้ว - รุ่น LT-32G33
(166 รีวิว)
4,990 บาท -29%
6,990 บาท
สมุทรปราการ/ Samut Prakan
Sharp HD LED TV ขนาด 32 นิ้ว รุ่น LC-32LE180M
(51 รีวิว)
5,639 บาท -29%
7,990 บาท
ตรัง/ Trang
Panasonic LED TV 40" รุ่น TH-40E400T
(71 รีวิว)
9,690 บาท -25%
12,990 บาท
การผ่อนชำระ: 10 x 969 บาท
สมุทรปราการ/ Samut Prakan
FLEDS LED TV 17 นิ้ว รุ่น TV17OE-D3 (Black)
(218 รีวิว)
1,740 บาท -30%
2,490 บาท
สมุทรปราการ/ Samut Prakan

Fig. 4. An Example of target document of the experiment

TABLE. II. EVALUATION RESULT



Evaluation Result of Scenario I				
URL Parameter	DOM extraction keywords	Target	Result	Invalid
http://www.lazada.co.th/shop-led-tv/	name	30	74	44
http://www.lazada.co.th/shop-led-tv/	name,image	30	58	28
http://www.lazada.co.th/shop-led-tv/	image,name,offers	30	30	0
http://www.lazada.co.th/shop-led-tv/	@type	30	30	0
http://www.lazada.co.th/shop-led-tv/	@type,image,name,offers	30	30	0
http://www.lazada.co.th/shop-monitors/	name	30	65	35
http://www.lazada.co.th/shop-monitors/	name,image	30	52	22
http://www.lazada.co.th/shop-monitors/	image,name,offers	30	30	0
http://www.lazada.co.th/shop-monitors/	@type	30	30	0
http://www.lazada.co.th/shop-monitors/	@type,image,name,offers	30	30	0
http://www.lazada.co.th/shop-kitchen-and-dining/	name	30	69	39
http://www.lazada.co.th/shop-kitchen-and-dining/	name,image	30	55	25
http://www.lazada.co.th/shop-kitchen-and-dining/	image,name,offers	30	30	0
http://www.lazada.co.th/shop-kitchen-and-dining/	@type	30	30	0
http://www.lazada.co.th/shop-kitchen-and-dining/	@type,image,name,offers	30	30	0

C. Result and Discussion

The experiment was designed to let our system extracts 30 product information from 3 different groups of the product represented by 3 URL Parameters. Each URL Parameter was tested with 5 different DOM extraction configurations. The result column represents the number of the record that is extracted and stored in the relational database. The extracted items were compared with original information to determine accuracy and invalidity, as displayed in the invalid column. The evaluation result is illustrated in TABLE. II.

The result shows that our system has successfully extracted the interested data from the target Web documents. However, invalid items were found when the DOM extraction configurations are not correctly set. We found that JSON data of the target Website is distributed in many locations of DOM. Consequently, the first and second try of our experiment has failed because the number of matching keywords in the extraction rule is not enough to distinguish the cached JSON data. Thus, selecting efficient keywords for DOM extraction rule is an important factor for our approach. Moreover, there are some considerations and limitations that should be discussed.

1) JavaScript page navigation: In some scenario, modern Javascript Web applications implement its navigation system using lazy loading technique. The first request will receive only first set of data. The remaining data will be fetched by clicking a button, which is sending another request to request next set of data from the back-end system. As a result, we have to add more URL parameter to the configuration to simulate a request that fetches least recent data. In order to automatically navigate through all data, this feature should be added to Extraction Engine.

2) Duplicate data: Since we cannot perfectly set the frequency of extraction to match the changes of data in the target Web sites, repeatedly extracting the same documents has caused duplicate records in both scenarios. Thus, we have added a configuration setting, i.e. "allow_dupplicate", to enable the user to specify wheater duplication is allow or not.

3) Limitation: Our proposed system is designed to extract the JSON data from the target Web documents. The target Web sites are required to expose JSON data in their DOM. However, to be seen and ranked by the search engines, some modern JavaScript applications render their pages from server-side as ordinary client-server Web applications. Thus, we believe that extracting data from DOM is still an important technique for the Web data extraction. As for the future work, we have planned to integrate state of the art DOM extraction techniques with our proposed method to allow universal Web data extraction.

V. CONCLUSION

This paper has presented a semi-automatic Web data extraction system that aims for extracting data from modern JavaScript Web applications. The proposed system enables users to select data from existing Web documents by defining extraction rules, schedules, storage and data transformation logic. The design of Web data extraction engine that utilizes the headless browser for fetching dynamic Web documents and the application of DOM parser techniques to extract the embed JSON data have presented and implemented as a command-line tool. The preliminary evaluation results have shown that our proposed system has successfully extract data from modern JavaScript Web applications i.e. online shopping. The limitations and future work have also been discussed.

REFERENCES

[1] C. Chang, M. Kayed, M. R. Girgis, K. F. Shaalan, C. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A Survey of Web Information Extraction Systems," IEEE Trans. Knowl. Data Eng., vol. 18, no. 10, pp. 1411–1428, 2006.

[2] E. Ferrara, G. Fiumara, and R. Baumgartner, "Web Data Extraction , Applications and Techniques : A Survey," ACM Trans. Comput. Log., vol. V, no. June, pp. 1–20, 2010.

[3] N. Negm, P. Elkafrawy, and A. B. Salem, "A Survey of Web Information Extraction Tools," Int. J. Comput. Appl., vol. 43, no. 7, pp. 975–8887, 2012.

[4] T. Gogar, O. Hubacek, and J. Sedivy, "Deep neural networks for web page information extraction," in IFIP Advances in Information and Communication Technology, 2016, vol. 475, pp. 154–163.

[5] D. Laishram and M. Sebastian, "Extraction of Web News from Web Pages Using a Ternary Tree Approach," in Proceedings - 2015 2nd IEEE International Conference on Advances in Computing and Communication Engineering, ICACCE 2015, 2015, pp. 628–633.

[6] S. López, J. Silva, and D. Insa, "Using the DOM Tree for Content Extraction," Electron. Proc. Theor. Comput. Sci., vol. 98, pp. 46–59, 2012.

[7] S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli, "Web wrapper induction: a brief survey," AI Commun., vol. 17, no. 2, pp. 57–61, 2004.

[8] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, "Single Page Application using AngularJS," in International Journal of Computer Science and Information Technologies, 2015, vol. 6, no. 3, pp. 2876–2879.

[9] PhantomJS, "Full web stack No browser required." [Online]. Available: http://phantomjs.org/.

[10] Lazada, "Lazada." [Online]. Available at: http://www.lazada.co.th/.