Bachelor Thesis Project

Scraping Dynamic Websites for Economical Data
- A Framework Approach
Abstract

Internet is a source of live data that is constantly updating with data of almost any field we can imagine. Having tools that can automatically detect these updates and can select that information that we are interested in are becoming of utmost importance nowadays. That is the reason why through this thesis we will focus on some economic websites, studying their structures and identifying a common type of website in this field: Dynamic Websites. Even when there are many tools that allow to extract information from the internet, not many tackle these kind of websites. For this reason we will study and implement some tools that allow the developers to address these pages from a different perspective.

Keywords: Web crawling, Dynamic Websites, Robots, Framework
## Contents

1 Introduction  
   1.1 Background ............................................. 1  
   1.2 Motivation .............................................. 1  
   1.3 Problem Formulation ................................... 2  
   1.4 Previous Research ...................................... 2  
   1.5 Research Question ...................................... 4  
   1.6 Scope/Limitation ........................................ 4  
   1.7 Target Group ............................................ 4  
   1.8 Outline .................................................. 5  

2 Different types of websites presenting economic data  
   2.1 Avanza ................................................. 6  
   2.2 MorningStar ............................................. 7  
   2.3 Fondmarknaden ......................................... 8  
   2.4 Refined Problem Formulation ........................... 9  

3 Method  
   3.1 Scientific Approach .................................... 10  
   3.2 Method Description ..................................... 10  
   3.3 Reliability and Validity ................................ 11  
      3.3.1 Ideal Approach ..................................... 11  
      3.3.2 Project’s Approach ................................ 12  
   3.4 Ethical Considerations ................................ 12  

4 Implementation  
   4.1 Tools and Technologies ................................. 13  
   4.2 Architecture ............................................ 13  
   4.3 WSSRobots Avanza ....................................... 16  
   4.4 WSSRobots MorningStar ................................ 17  
   4.5 WSSRobots Fondmarknaden .............................. 19  
   4.6 Framework Generalization .............................. 21  
   4.7 Problems ............................................... 24  

5 Results ....................................................... 27  

6 Analysis ...................................................... 29  

7 Discussion .................................................. 30  

8 Summary, Conclusion and Future Work  
   8.1 Future Research ......................................... 31  

References ................................................... 33
1 Introduction

In traditional web applications, each page has a unique URL that refers to it. But not all web applications are structured like this. There are a number of websites that are built as AJAX applications, where each state is not represented by a unique URL. In these kind of applications content can be loaded dynamically while surfing through different states of the same web page using JavaScript, and this make most of crawlers not to work properly with them.

In this thesis we will explore the characteristics and how to build a crawler for these specific scenarios, as well as the problems it has to face in order to be able to extract the desired information from the dynamic website.

1.1 Background

Newspapers are publishing new articles 24 hours a day on the web. Keeping track of those articles can be easy if it is only from one source, but when it comes to the Internet, there are too many sources to make this an easy task. There are plenty of blogs, newspapers and web pages that publish new articles quite often. In general, many people just open each single site in order to get updated on those articles. This is a very time-consuming task that could be easily improved. There are already some tools that make this task easier, as RSS readers, but those only involve blogs, not web sites or online newspapers. This problem involves backgrounds as data mining and web crawling, as well as how to implement the architecture for a piece of software that has many subsystems.

Data mining and web crawling are two related main subjects that are quite important nowadays. While web crawling relies on finding new sources and web pages that are published on the Internet, data mining focus on extracting the information that they contain. Most of the web crawling challenges have been already solved, but as the nature of the Internet is to constantly evolve and update, there is an imperious need of keep improving these techniques in order to deal with new challenges.

In the last years, AJAX (Asynchronous JavaScript and XML) has gained a prominent position with Web 2.0 [1]. This technology brings to the websites a much more dynamic interaction where users do not need to go through different pages in order to reach their goal. AJAX uses JavaScript in order to dynamically load new content as requested in a web page, avoiding the previous need of refreshing or going to another page. This is a great improvement over previous solutions, but also comes with its own challenges.

Through this thesis we will address the challenges that come with AJAX websites and the possibility to crawl and extract specific information from them.

1.2 Motivation

In order to motivate this project, an introduction of LnuDSC is needed. LnuDSC (Lnu Data Stream Center) is a project presented by professor Jonas Lundberg which purpose is to access to live website data. This project is expected to have many different parts and to be quite complex. For this tool to be possible, there is the need of having several scraping tools that monitors the specific websites 24 hours per day, gathering all the new information and sending it to the server. Those scraping tools are the so called Robots.

Facing the problems of this project, one of them is that each website is unique, and consequently, each website would need the implementation of an specific robot that gathers its information. For implementing this robot, there is a need of understanding and knowing the specific structure of each website, as the information shown there need to
be accessed. As professor Lundberg estimates, sometimes that is an easy task: “Twitter: about 6 hours”, but in other cases it can be really complex: “Avanza: no success after a week”.

Within this project, we believe that every open website can be scraped, and the purpose of LnuDSC is to provide a framework that makes this process much easier and available for end users that want to consume live website information.

LnuDSC is a multidisciplinary project that would involve several people from different subject areas, so throughout this project we will try to contribute to it with a specific part involving the robots and scraping tools for specific scenarios.

In the following chapters the specific problem will be determined. We will also give an overview of this LnuDSC project architecture as well as specify where this thesis will specifically be placed inside the project.

1.3 Problem Formulation

The problem we will try to solve through this thesis is part of the bigger project LnuDSC that consists in developing a system that provides to the final users specific content from different websites automatically. In order to achieve that, this project has been divided in different sub-projects due to the difficulty of some of them. While in the whole system there is the need of a database, the client for the final user, a server where all the software runs and the communication protocols between all the modules, in this thesis we will focus on how to build the robot that extracts information in a specific case that has been identified as a problem and complex part.

As this project wants to retrieve information from many different websites, these can have quite distinct structures and characteristics. While looking at some of this pages and focusing on the task of extracting specific information from them, we can identify at least two kinds of websites: ASP or AJAX based and others more generic and static. As we mentioned before, there is plenty of work done in data and information extraction from different websites, but most of the available tools and researches are focused on the most generic websites. In order to crawl and get the information of ASP or AJAX websites, few tools have been developed and it has become also a subject of research due to its complexity.

For this thesis, the main problem we will try to solve is centred on developing a robot that can extract information from some specific AJAX or ASP websites and later on, trying to get a more generalized form of this tool that can be used for other AJAX pages. The problem with these dynamic websites resides on its nature. In order to make these web pages more interactive, the content of the page is not completely loaded once we go to its main page. While generic websites can be crawled so for each entry or useful information we can find a specific URL, that is not the case with AJAX based sites. In these, all the content is under the same page, but in order to have access to it, the user has to interact with that page. This is a non-trivial task for automatic programs as crawlers or extractors, that don’t have the needed information to interact with a website as an end user.

1.4 Previous Research

The topic addressed by this thesis has already been studied by many other researchers as it is a really important issue faced by many companies and institutes. As Gupta mentions, web crawlers are almost as old as the web itself [2]. Since the constant evolution and the increasingly number of pages on the Internet, the necessity of automatic programs that
could crawl and extract information from the web existed. Also, as Rajapriya says, search engines became a need and, as a core part of them, crawlers had to be implemented and constantly go through all the possible web in order to gather all the needed information to make those search engines possible [3]. Another use mentioned by this author is in data mining, where these crawlers have to extract and analyse information for statistical purposes. Also it is mentioned some of the problem that have been faced while implementing crawlers, as for example finding duplicated URLs. As it is shown, many research has been made in this area due to its uncountable possible uses. Also due to the huge diversity of the web there are many different implementations. On the Internet there are a huge amount of data represented in so many different ways and with many different structures. In order to be able to crawl the most of the data, it is a need to understand and to study all these kind of structures and the possibility of crawling them and extracting information from them. As it is known, the web is constantly evolving and new paradigms and challenges can be found. As it is mentioned by Kim, there was a time when most of the content of the web was written in markup languages as HTML, but nowadays the prominence of the Web 2.0 and the semantic web has brought new languages as RDF and OWL [4], languages providing dynamic websites, which mean new challenges and new researches for the crawlers and extractors.

It is also important to mention several tools that also address the problem of crawling and extracting information from different websites, as well as why those tools are not valid for the task presented within this thesis. Most of the tools that can be found on internet are able to extract data for the type of websites that we called "static", those ones that have a unique URL for each piece of information or for each entry. In order to support our research and to understand the current limitations of the available software, many tools have being tested on some extent. Due to some limitations, as some software not being completely free or having only the possibility to try the "Free-trial" version, we could not do an analysis as deep as we would like to, but is enough considering it as auxiliary and supporting material. Below we will list some of them that have been considered quite popular:

- **Scrapy**: This is a free and open source web crawling framework for python. It can be used for scraping or for general crawling. Within this project, it would have the disadvantage of using python instead of Java, which is the language used for the whole LnuDSC project. Also, it is not clear if it supports AJAX or dynamic websites in order to extract its information. While testing this application, it offers plenty of possibilities and tools. Through an application implemented using this framework it was only possible to extract the first page of the dynamic websites, and from them it is perfectly possible to extract the desired information available in that page. However, this framework does not provide any method to interact with the website, which is needed in order to retrieve the dynamic information that cannot be found in the first page of the website. In the official site more information can be found [5].

- **Lexalytics**: Lexalytics is not exactly a crawling tool, but a tool that is able to extract information and to analyze the text inside some websites, so it can also be related to the problem we are addressing through this project. This tool provides an API that allows the developer to call different methods in order to extract different information from a website. Nevertheless, the reason why this tool is not appropriate for the robots presented on this thesis is that it is more oriented to analyze blogs or text rather than extract some specific information like economic data. It is also im-
important to mention that it is not a free tool. While trying this tool, it was not possible to extract the information from any of the pages of a dynamic website, as this tool works over plain text in order to analyze it. It has some utilities in order to parse simple schemas of websites and extract its plain text, but this is not compatible with a dynamic website where its main content can be found in tables rather than in full paragraphs of plain text. For further information the official website can be visited [6].

- Import.io: This tool, as well as lexalytics, is not free (even though they have free plans depending on the amount of information extracted). In opposition to the previous tools presented, import.io provides its own API and is not intended for pure developers, but for being used as an standalone application that already extracts the desired data. Is this one the main reason why it is not an appropriate tool for this project, including also that the possibilities to extract data from dynamic or AJAX websites was not found. During our evaluations, this tool had a structured outcome providing and extracting all the information presented on the first page of the dynamic website, with the possibility of having this information extracted in a CSV file. Nevertheless, it was also not possible to interact with the website in order to obtain the unloaded information from the following pages, which made this tool not to fit the requirements needed in order to fulfil and to answer the needs of this thesis. More information of the tool can be found at its official site [7].

### 1.5 Research Question

This thesis wants to contribute to the development of a system that provides online information from specific websites. In order to achieve that, we want to focus on exploring the possibilities of extracting information and crawling these non completely loaded pages or non trivial sites where it is not possible to find all the information needed without interacting with the page. For this reason, we will establish the following research questions:

<table>
<thead>
<tr>
<th>Main RQ1</th>
<th>How to implement a Web Spider, Crawler and Extractor that automatically retrieves information from dynamic websites?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main RQ2</td>
<td>How to implement a framework that help to speed up the development of robots that automatically retrieves information from non-trivial or non completely loaded websites?</td>
</tr>
</tbody>
</table>

### 1.6 Scope/Limitation

In order to stablish a scope to this project, it will be referred after refining the problem formulation in the chapter 2.4. There the limitations of this thesis will be addressed with a better understanding of the given problem and the expected results.

### 1.7 Target Group

As the purpose of this thesis is to develop a framework that speeds up the implementation of specific robots, the target group of this thesis are those programmers responsible for developing web scraping robots. Within this project we will explain and provide algorithms, tools, examples and a framework that will help anyone with the will of implementing a robot that extracts economic data. In general, there will be programmers and developers related to data mining, web mining and web scraping. It could also apply for some
economists with some basic programming knowledge that want to implement a tool that keeps them up to date with some interesting economic websites.

1.8 Outline

Throughout this thesis we will, first of all, study the different type of websites that present economic data in order to understand in which context this project is focused on. We will study at least 3 different websites that present economic data: Avanza, MorningStar and Fondmarknaden.

In the next chapter, we will explain the followed methodology for this thesis, explaining the reasons why we chose the specific approaches and also considering some possible ethical issues according to this project.

In the fourth chapter, an explanation of the implementation process is provided for all the tools and steps implemented. We will give an overview of the involved tools used for this project, the architecture of our whole project and the specific implementations for the specific robots. Afterwards the framework generalization will be explained, pointing out some of the found drawbacks in this implementation.

In the final chapters, we will present the obtained results as well as an analysis of them. We will also discuss those results with the previous research given.

Finally, we will present some conclusions and point out some of the possible future works that could be done with this thesis.
2 Different types of websites presenting economic data

In this chapter we will introduce the general structure of different websites presenting economic data. We will focus on fund prices, although a similar approach would also work on stock market prices and other economical instruments. A fund is a sum of money saved or made available for a particular purpose[8]. For our purposes, these funds in most of the visited websites get updated once every day. Through this description we will try to explain which are the challenges and how the websites look like, which is important in order to be able to extract its data in the following chapters.

2.1 Avanza

Bankaktiebolaget Avanza is the largest online stock broker in Sweden with more than 400,000 customers and the largest number of deals on the Stockholm Stock Exchange [9]. Below it is shown the structure of its table and how the funds information is presented:

![Figure 2.1: Table containing the funds that the application needs to extract in Avanza website.](image)

This website provides a table where all the information of the funds can be accessed. As it is shown in the bottom part of the previous image, some navigation buttons are presented. As an important characteristic of the website, in order to navigate through the different pages of the table there are different options (that are related). Once the user click on the "next" button, the URL of the site is updated. This detail will be of crucial importance when we describe the process of crawling the website and extracting its information. Avanza.se can not be considered a page that dynamically updates its content, as
it provides an specific URL for each page of the table, so every piece of information can be accessed through a different unique identifier or URL.

In this website, each fund gets updated generally once every day, and as May 15, Avanza had 1293 funds distributed across 44 pages.

### 2.2 MorningStar

MorningStar is a website with some specific problems when it comes to be crawled. The main section that was of interest for this project was retrieving the information of some funds as it is shown below:

![Figure 2.2: Table containing the funds that the application needs to extract in MorningStar website.](image)

In this screenshot we can see the main structure of interest in the website. The purpose of the application is to be able to read each row of the table, parse the information found and send it to the server. Here the main problem is the way this table is implemented and shown. In the bottom of the image we can see how this table can be navigated, going from one page to another through these navigation buttons. The main difference with other more general websites is that those navigation buttons are the only way to move through pages in the table, using AJAX. In other approaches, the action of moving from one page to another would be reflected on the URL as well, making it much easier to be
crawled or its information to be extracted.

On May 15, MorningStar.se had 21085 funds distributed across 1055 pages.

2.3 Fondmarknaden

Fondmarknaden is, as well as the previous MorningStar, a website with investment and funds information. When it comes to the structure of this site, it is quite similar to MorningStar, containing all the funds information inside a table that can be navigated using AJAX. In the following figure it is shown the similarity of the structure of the main table, even when the style of it might be different:

![Table containing funds information from Fondmarknaden](image)

Figure 2.3: Table containing the funds that the application needs to extract in *Fondmarknaden* website.

Here it is shown the table that the application has to crawl and the information contained. As mentioned before, this table and its structure is very similar to the previous one of MorningStar, and this is one of the main reasons why this website was chosen, but still has some small differences that must be taken into consideration when implementing the application.

As it is such a similar website, all that was said for MorningStar is also valid for Fondmarknaden. The interaction needed with the website is exactly the same and the limitations and problems found here as well.

However, it is important to mention the style differences between both websites, and in the following figures it is shown the comparison between a generic row from both websites:
As it is shown, both rows have some differences that are really important while extracting its information. In the row of MorningStar, there are two cells with different elements before the title of the fund, while in the Fondmarknaden row, the title is in the first cell of the row. If we follow this process, it is shown how the cell and the position where each piece of information appears is different.

On May 15, Fondmarknaden had 1705 funds distributed across 57 pages.

2.4 Refined Problem Formulation

As we have selected and defined some websites that provide economic data, now it is time to divide them and explain the main differences between them. As we mentioned, Avanza is an investment website that provides all the information within a table that can be navigated directly from the URL, while MorningStar and Fondmarknaden can not. In order to navigate these two websites AJAX is used, which means that they are dynamically updated depending on the interaction that the user has with the website, for example, while clicking on the "next" button.

Given that, in a first step, we will focus on extracting information for the website avanza.se [10], which is a more general and easier kind of website to crawl and extract information from. In a second step, the more complex websites will be crawled, trying to extract its information through the tool. Here the purpose is to monitor each fund in the websites, aiming to present any price update of each and every fund as soon as they appear on the website. For this purpose, we selected the websites morningstar.se [11] and fondmarknaden.se [12], due to the complexity of them and its characteristics. It is important to mention that, while they share the way of updating its content and how their tables can be navigated (where all the economic information can be found), their style and some details are slightly different.

After creating tools that are able to extract that information, a generalization will be made, trying to implement a framework that allows developers to create new robots for other similar economic websites in a much easier and faster way.

This will be the scope of the thesis and the mentioned websites as well as the problems that will be mentioned in the following chapters will be the limitations that we will set.
3 Method

In this chapter we will describe the different scientific approaches that we followed in order to answer our research questions.

3.1 Scientific Approach

The selected research approach for this thesis will involve mostly qualitative methods, focusing on the "Action Research" methodology. As mentioned in the problem formulation, we will try to implement a working robot and a framework to develop specific robots. Gathering numerical data from this kind of implementation would be more complicated and not give relevant data for the mentioned purpose. Nevertheless, some quantitative data will be provided through the results, as the number of lines needed for the implementation or the needed time to run the experiments.

In a first step, the analysis of the different websites is needed. There we will need to ask exploratory questions, using a "Description and Classification" methodology. Already in the previous chapter we presented the different websites under study, where it can be found their description and categorization according this method. After having this overview and describing the websites, we will need to have a deeper understanding of them. Following a "Descriptive-Process" methodology, we will try to understand how the websites work. This will be needed in order to provide a valid answer to our research question.

Finally, the best way to give an answer to the research question will be using design questions, checking which are the possible ways or how to design the robot and framework that can solve the mentioned problem.

3.2 Method Description

Once presented the different research methods that will be followed, here we will specify the way that they will be applied.

In order to implement this robot and framework, an iterative and cyclic methodology will be followed. This means that in a first step we will try to develop a working robot that is able to crawl a specific website. Afterwards we will check possible errors and fix them, introducing new code from the results obtained previously. The next step will be to replicate this process and implement another robot for a different website. Finally, we will try to generalize this implementation extracting common steps and code, obtaining from them a framework.

In order to analyse the different websites, a simple comparison and description will be provided. We will visualize and provide screenshots of the website and describe what it is shown on them, focusing on the specific parts that are of importance for this thesis. Also they are categorized under the “dynamic” or “non-dynamic” categories, as those are of main importance for the purpose of this project.

Later on, we will study how the website works. In order to achieve that, we will inspect its available source code (HTML and JavaScript mainly), trying to understand which processes the website is following to provide specific data that it is of interest for this project. In following chapters some screenshots of this inspection will be shown, helping to understand some of the main processes of the studied websites. Finally, in order to answer the research question, we will provide a description of the implementation. Here we might find two approaches: how does the application work and how does the design work. As mentioned in the research question, the main purpose is to design a robot and a
framework that can deal with specific websites, so here we don’t consider its performance as the most important data we can provide. However, some of this data can be provided, but it will be more important that data that explains the implementation itself and how it can be replicated in other scenarios.

3.3 Reliability and Validity

Here we will discuss the reliability and validity of this project. We will analyze the ideal approach of how this should be conducted and also the taken approach due to time and resource limitations.

3.3.1 Ideal Approach

In order to have valid and reliable data, a whole process of verification and study of the tool should be made. First of all, the tool should be tested repeatedly under the same conditions, considering same time frames, same internet connection and same working load on the computer that runs the program. Once done this, the obtained data should be compared both between them and also with the raw data of the website. This would verify the correctness of the extracted data by the implemented crawlers.

By this approach, we would be able to state the reliability and validity of this tool, as well as its drawbacks. However, some of these conditions are really difficult to repeat as we do not have the resources to control them. For example, we are not able to have complete control on the working load of the computer in two different runs of the application, as many system processes might change without we noticing. Also the internet connection, or more precisely the whole connection and load of the server in order to have control of the response times, is something impossible to completely control.

Also referring to the obtained data, in this case is also difficult to compare it as it should constantly change (the funds are updating in general every day), and manually comparing all the data with the website itself is quite time-consuming that is not reachable for this project.

Focusing on the framework implementation and its usability, which is the main part of this thesis, two different groups of programmers and developers should be set. Once we have these two groups, a set of tasks should be given. The tasks would consist on implementing a robot that can extract the desired information from at least two different economic websites. Also, this robot should be implemented twice: once using the framework and the other time without using it. Also in order to avoid some correlation while implementing a robot for the same website with different tools, it would be interesting to change the order of the implementation for both websites. So the test would be as follows:

1. Implement a robot for website 1 without the given framework.
2. Implement a robot for website 1 using the framework.
3. Implement a robot for website 2 using the framework.
4. Implement a robot for website 2 without the given framework.

Once done this, we should statistically analyse the needed times for each task, getting then some interesting and reliable results. It is also important to mention that those results would be more reliable the more programmers would get involved in the test, but with two or three programmers we would already have some reliable results.
Because of these reasons, this ideal approach can not be taken within this thesis, so we will take a less reliable or valid approach but that can give us some insights and an idea of how it works.

### 3.3.2 Project’s Approach

In order to verify its correct functionality, we chose to make this application run for over 24 hours, retrieving and updating any possible fund. Doing that, it is possible to verify its correct functioning as well as to find possible errors that might occur.

Once obtained the data, it is needed to take a look both to the data and to the console log in order to verify that the data is correct. In order to do so, as the amount of data is too big to validate it all, we will just take an overview, focusing on where some problems might be expected, but also understanding that if the application works well for an amount of data, it can be considered that there should not be any further problems. This is because the websites maintain a common structure along their many pages, so we can expect the application to work the same way with one page and with the rest of them. With these steps a manual verification of the correctness of the data has been done.

For the final step in order to evaluate the validity of the framework approach, we will compare the needed work while developing the robots without using the framework and then using it. In order to compare the needed work, time and number of lines of code will be considered.

### 3.4 Ethical Considerations

Crawling and extracting information from the Internet is quite a common practice that is being done by many companies all around the world. It has a lot of obvious benefits that have been already discussed, but they do not come with some ethical and even legal drawbacks.

Most of the websites in the Internet are public, which means that anyone is allowed to enter and consume its content, but also many of them keep some copyrights that are not always respected while crawling or extracting data with automatic robots as the one that is being implemented through this thesis.

This is a topic that depends a lot on the kind of content that different websites might provide and how the information extracted from those will be used. For example, there are some specific tools called "aggregators" that provide content extracted from other different websites. Those aggregators can be really useful, but using them means not using the official websites that provide this information. Those original websites might get benefits from the number of visits their website has. Taking these two facts into account, it can conform an ethical and, in some situations, even a legal problem.

A well known issue related to this are the news aggregators, that have been forbidden (or that have to pay some taxes) in some countries. For example, recently Google News, a very well known new aggregator, decided to close and not provide this tool to some countries where these kind of tools, due to the problem mentioned before, were asked to pay a tax [13].
4 Implementation

Through this section we will introduce and discuss all the details of the implemented application, listing the tools and technologies used, its structure and architecture as well as its design. Here some decisions will also be argued. Finally we will discuss some of the problems found during the implementation and explain how they were solved.

4.1 Tools and Technologies

For developing this applications the following technologies and tools have been used:

- IntelIJ Idea 15.0.4 Student Version
- HtmlUnit 2.20
- Java JDK 1.8
- JSOUP 1.8.3

IntelIJ Idea [14] is a very popular IDE (Integrated Development Environment) for Java created by JetBrains. I chose this IDE instead of Eclipse (probably the most popular IDE for Java) due to its performance and the experience I previously had with it.

As an important part of this implementation and a result from the previous research done, HtmlUnit was chosen and it is a central tool of the implementation of this thesis. HtmlUnit is a “GUI-Less browser for Java programs” [15], that models HTML documents and provides an API that allows to invoke pages, fill out forms, click links, etc... This tool is of high importance in this thesis due to the nature of the AJAX applications, that need user interaction in order to display all its content. This tool declares to have good JavaScript support and to be able to work with quite complex AJAX libraries (even when it is not limited to those). It can also simulate different browsers as Chrome, Firefox or Internet Explorer depending on the configuration used. HtmlUnit is generally used for testing purposes, but in this situation and as a result of our previous research, it will be used in order to retrieve information from complex and not completely loaded AJAX websites. Further information of the tool can be found on its website.

The selected JDK (Java Development Kit) [16] was the latest release by Oracle at the time this thesis started. This makes this thesis being using the most updated tools and as Java has included and changed quite a lot through its different versions, we ensure that it is compatible with the latest technologies.

Finally, jsoup is a Java library for working with real-world HTML [17]. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods [ jsoup.org ]. This library is used where HtmlUnit is not needed, as they provide similar tools for the same elements through the DOM. In this specific project, it is used for the avanza website, as it is not dynamically updated so it does not need of HtmlUnit to be crawled.

4.2 Architecture

LnuDSC is the project where this thesis is focused on. This project consists on providing a tool that automatically detects updates on specific information in some web sites and
retrieves this information to the final user. Due to the complexity of this problem, it has been divided in different sub-projects where each focus on a different subproblem. In order to have an overview of the problem, the following diagram is provided:

**Figure 4.6: Diagram of the main system LnuDSC.**

In the figure it is shown an overview of the architecture of the whole system. On the left, the end user will be able to connect and interact with the system through a client. This one will be connected to the main server, from where it will get all the needed information. This server will have a database where all the updated information will be stored, and it will also have several robots that will be the ones that constantly crawl the web in order to find new content. Is in this last part where this thesis will focus, implementing one of the robots for a specific and complex scenario. In the following diagram it is shown the main structure of the WSSRobot, where this thesis is focused on:

**Figure 4.7: Diagram of the involved classes in a WSSRobot.**

Now focusing on these Robots, it is important to define their structure and how they have been implemented. For this thesis, two funds websites have been selected to implement a robot for each of them, trying to solve its problem, find common steps and
procedures and pointing out the difference between both applications. In order to do so, some basic classes that represent some needs on the implementation have been found:

- FakeJsonConsumer
- Fund
- JsonObject
- Util

Each of this classes (or interfaces) represent an object that will be involved while extracting information from what is expected to be any website with similar characteristics. At the same time, is a way to establish some common classes and interfaces that provide a protocol of communication between the robots and the servers. Below a UML class diagram of these basic objects is shown:

![Class Diagram](image)

**Figure 4.8:** Class diagram of the basic needed classes involved in WSSRobots.

First of all, the *FakeJsonConsumer* is a class that simulates the behaviour of the client that will connect to a server. It receives the json messages sent by the different robots and print on a file the funds that have been extracted or updated. Second, a *Util* class is provided, containing some common methods used, as for example *addToQueue()* , which is used while sending the JSON messages from the robots to the client (or in this specific case, to the *FakeJsonConsumer*, which simulates the behaviour of the client). Then the interface *JsonObject* has been implemented in order to provide a common method *toJSONString()* , which is necessary for the communication between the different parts of the whole project. Finally, the *Fund* class implements the previous *JsonObject* interface, and it is an object where the data of the fund extracted is stored and parsed. Some specific
information of the funds have been pointed out as for example currency, date, price or title, and the within the application we fill those fields creating different Fund objects.

In the following figure it is shown an example of a real Fund obtained from the Avanza website.

```
{
    "FundName": "Swedbank Robur Access Mix",
    "fundUrl": "https://www.avanza.se/fonder/on-fonden.html/206/swedbank-robur-access-mix",
    "currentPrice": 236.27,
    "discoveryTime": 1464063440887,
    "datum": ",",
    "id": 206,
    "currencyCode": "SEK"
}
```

Figure 4.9: Example of a JSON fund object markup.

As it is shown, we can appreciate the structure of this JSON object. In this case the only field that could not be obtained is the "datum". On the other fields we can see all the needed information.

In the following subsections, the different specific implementations for the different robots will be presented.

### 4.3 WSSRobots Avanza

As it was mentioned before, avanza website can be considered as one of those which are not dynamically updated. For that reason, the way to crawl and extract information from this website is easier and does not involve some of the tools that are mentioned above. In order to navigate the different pages from this website, it is only needed to request the specific URL for each page. For doing that, a counter is implemented, which goes from the first page selecting the number 1, until the last page, modifying only this number in the URL. Once the specific page is returned, to parse its information is the only thing left.

In order to parse the information of each table’s page, JSOUP has been used. This library allowed us to parse the table using the DOM and getting each element by different characteristics as its value, id, etc.

Once the information have been extracted, it is stored as JSON and sent to the FakeJSONConsumer.

To have a better understanding of this process, an algorithm will be provided. This algorithm is specific for the Avanza website, but can be used for the other scenarios as well, as the differences with the other websites resides on how to implement the algorithm, not in the algorithm itself. Below the algorithm is provided:
Algorithm 1 Avanza robot algorithm

1: procedure INITIALIZATION
2: Initialize a JSON Queue
3: Start the FakeJsonConsumer
4: Start the Robot/Crawler
5: procedure INITIALIZE ALL FUNDS
6: oldFunds ← traverseAllFunds()
7: function TRAVERSE ALL FUNDS
8: page ← website(url)
9: fund ← extractFund(page)
10: queue.add(fund)
11: while true do
12: procedure UPDATE FUNDS
13: allFunds ← traverseAllFunds()
14: if \( \exists \text{fund} \in \text{allFunds} \neq \text{fund} \in \text{oldFunds} \) then
15: queue.add(newFund)

In the previous algorithm it is shown the different stages of the application. In a first step, the robot, through its Main class, initializes its components creating a JSON Queue and starting both, the FakeJsonConsumer and the Robot/Crawler itself. In the second step, we initialize all the funds, running for the first time the method traverseAllFunds() and storing its result. Finally, we enter a loop that searches for updates running the same traverseAllFunds() method and comparing its results with the ones obtained previously, detecting then any possible update.

4.4 WSSRobots MorningStar

Previously we mentioned the structure of this website and some of its problems when it comes to be crawled. Now in this section we will focus on how the specific robot for this website was implemented.

As we have to interact with the website in order to navigate from one page to another, in the implemented application this interaction was simulated using HtmlUnit. Creating a GUI-less browser helped to select and be able to click the navigation buttons of the table. Nevertheless, in order to do that we need some previous knowledge of the page, inspecting its HTML code. Through this inspection, we will need to identify the different parts where we are interested. HtmlUnit can use either id or different kind of paths, as for example XPath (being this one our choice) in order to access and interact with the different buttons.

Bellow we can see a piece of code that shows how the action of clicking on the "next" button is implemented:
import com.gargoylesoftware.htmlunit.WebClient;
...

WebClient webClient = new WebClient();
HtmlPage page = webClient.getPage(url);
while (page != null) {
    // XPathNext will be obtained by Inspect Element
    HtmlAnchor next = (HtmlAnchor) page.getByXPath(XPathNext);
    page = next.click();
}

In the previous piece of code it is shown how to perform a "click" on a button as an example. First of all, we need to have the WebClient initialized, as it is the main object needed to do any action with HtmlUnit. With this object, we obtain the page of the website. Once having the page, we must identify the "next" button. In this code, we do this through its XPath, that should be obtained through a previous inspection of the website. Finally, we can use the click() method provided by the tool, which will retrieve the new page.

In the following image it is shown how, through "Inspect Element" in Google Chrome, we can access to the HTML code of the mentioned button we want to interact with, obtaining the needed information that HtmlUnit needs, in this case, the XPath.

Figure 4.10: Inspect element selecting the ">" button on the MorningStar table, using Google Chrome.

Once we have these information, we can introduce them to the application, which will extract the information with the given paths and ids, and will navigate through the different pages of the table clicking on the specific buttons we select and loading the different pages.
Taking a more detailed view of how the code of this application works, in a first step the HtmlUnit WebClient is initialized. After, the first thing is to obtain the home page where the first table’s page of interest is. For that the specific URL of the website is used. From there, we start to select the elements of interest, as the table itself, the different rows, the “next” button and in this case an extra step that is selecting a specific tab of the table. Once all the elements are correctly obtained and selected, the loop that parses the desired information from the different rows start. Within this loop, we check whether the information is already extracted or not, acting consequently in order to store or update these data. After the loop reaches its end, when there are no more rows to parse in the specific page, we simulate a click on the selected “next” button to get a new page of the table where more rows can be found. From here on, the process is repeated. Once all pages of the table have been visited, we restart the HtmlUnit WebClient in order to solve some memory problems that will be referred afterwards and to clear the unused memory and the application starts crawling the whole table again searching for updated information.

It is important to mention that within this robot the different information is kept as caché, due to the purpose of the whole project. All the information gathered by the robot will be stored afterwards in a server that is not implemented yet and that doesn’t belong to this subproject, so in order to solve this, we keep all the extracted information in local variables while sending them in JSON format to an arbitrary JSON consumer.

4.5 WSSRobots Fondmarknaden

In order to crawl and extract information from this website, all the needed ids and paths will be taken, just as it was done previously with MorningStar. Below it is shown the same process of inspect element in order to obtain the XPath of the "next" button:

![Inspect element selecting the "Nästa" button on the Fondmarknaden table, using Google Chrome.](image-url)
In order to have a better understanding on how these robots work, we will provide a more specific algorithm on the navigation of the websites using the paths obtained by the element inspection:

**Algorithm 2 MorningStar/Fondmarknaden navigation algorithm**

1: \( \text{page} \leftarrow \text{webClient.getPage(url)} \)
2: \( \text{while } !\text{stop do} \)
3: \( \text{table} \leftarrow \text{getTable(tablePath)} \)
4: \( \text{rows} \leftarrow \text{getRows(table)} \)
5: \( \text{for } \text{row} \in \text{rows do} \)
6: \( \text{fund} \leftarrow \text{parseFund(row)} \)
7: \( \text{if } \text{fund} \text{ not in allFunds then} \)
8: \( \quad \text{allFunds.add(fund)} \)
9: \( \text{else fund is updated} \)
10: \( \quad \text{allFunds.update(fund)} \)
11: \( \quad \text{queue.add(fund)} \)
12: \( \text{if next } \leftarrow \text{page.getNext()} \text{ then} \)
13: \( \quad \text{page } \leftarrow \text{next.click()} \)
14: \( \text{else} \)
15: \( \quad \text{stop } \leftarrow \text{true} \)

In this algorithm it is shown the process of how we navigate through the different pages in these dynamic websites. The first step is to get the first page. For that, it is only needed to call the `getPage()` method with the given URL. It is after this when we start a conditionally loop that will keep running until we find a hint saying that the actual page is the last one. Inside this loop, we extract the table through its path (or id) that has been obtained through previous inspection of the website. From this table, all its rows are obtained. Those rows are the different obtained funds, but in order to extract them, we need to parse those rows with the method `parseFund()`. Then we check for possible updates of this fund, adding it if it is a new fund, updating it if it has been modified or doing nothing in other case. The last step is finding the next button again on the current page. If it can be fund, it is clicked and retrieved a new page, starting the loop again. If the next button can’t be found, the loop ends here.

Once these information is collected, we follow the same steps as with the MorningStar Robot, changing only those details where both websites differ.

As the code and how it works is exactly the same as with the previous robot, here we will mention some of the main differences between them. As it was said, the first step is changing the ids and paths to the ones of this website, that are completely different just because of its HTML structure. Obviously, the URL that is introduced to obtain the home page is different, as it is another website the one that we are extracting information from. Furthermore, in Fondmarknaden it is not needed to press on any tab of the table, as all the needed information is provided already in the main tab. Also in this table, the number of rows per page ascend up to 30, so we change that parameter as well as an indicator of the last page of the table. Finally, and as a parameter that will take importance in the generalization of this application, it is needed to redefine the position of the information we want to extract inside the row of the table.
As mentioned and compared before, MorningStar and Fondmarknaden have a lot of similarities, but inside the rows of their tables, where the funds information is, we must consider its differences and they have to be reflected in the code. In order to do that, a method that parse each row or fund has been implemented, introducing there the specific position where the information such as title, price or currency are located.

Once all this information is provided and filled in the program, the configuration and the behaviour of the application is identical to the previous one.

4.6 Framework Generalization

The main purpose of this project is to implement a tool that can crawl and extract data from many different websites. As it has been discussed through the previous sections, the problem that is been faced with AJAX websites makes it very difficult to develop a unique tool that is able to do so. Instead, and in order to solve this problem, a framework has been implemented as an intermediate solution that provides enough implementation to make it much easier the task of implementing robots that can crawl many different AJAX websites with a similar structure as the ones implemented for this thesis.

In order to implement this framework, the process has been to generalize the solutions that were implemented for the specific scenarios discussed above; MorningStar and Fondmarknaden. Taking a look to both specific implementations, it is easy to see a lot of similarities. Almost all the process and the steps followed have been the same, and finding the differences between them allow us to develop a more generalized framework that can be used to implement extra robots.

As mentioned before, this framework is based on MorningStar and Fondmarknaden, leaving Avanza behind. This is due to the differences explained before between these two kind of websites, being the first group the dynamically updated kind of websites we are focusing on. This is the main reason why Avanza is not part of the implemented framework.

In order to have a deeper understanding on how this works, we will list and describe the function of the main classes implemented for this purpose:

- Main
- FundCrawlerFramework
- Spec
- Implementation using Spec
Going in detail into the code, the different classes listed above will be explained. First, a Main class, that will be only the one who calls the application developed with the framework. Bellow it will be shown how the code of this class looks like. Second, a public class FundCrawlerFramework, that include all the common steps and processes followed in order to crawl and extract data from the different websites, making it general enough to be able to adapt to different scenarios depending on the parameters. The third class implemented is an abstract class, Spec, which contains all the specific methods that must be implemented while using this framework. In order to select and define those methods, the specific steps that are needed for each website have been pointed. This makes that, for the implementation of an specific tool using this framework, it is needed to study the specific scenario and implement the specific methods included in this abstract class in order to successfully extract the desired data from any AJAX website with a similar structure as discussed above.

```java
public class Main {
    public static void main(String[] args) {

        BlockingQueue<JsonObject> producer2consumer = new LinkedBlockingQueue<JsonObject>();
        FakeJsonConsumer consumer = new FakeJsonConsumer(producer2consumer);
        consumer.start();

        String website = ""; // Website URL
        String tableId = ""; // Id of the table
        String nextBtnPath = ""; // XPath of the next button
        int numRows = 20; // Number of rows in the table
```
A more detailed explanation is needed when referring to the `FundCrawlerFramework` class, as it is the one that englobes all the common methods and that was extracted from the previous implementations. In order to make it a framework, it was needed to understand and separate every step that could be divided in order to make it more modular. That is the reason why many new methods can be found in this class that couldn’t be found in the previous implementations. All these methods provide a functionality that, during the analysis of the code and the websites, was understood as a separate step that could be taken in a different way in different websites with other contexts, but that in the economic data websites we studied were the same. That helped to identify as well those steps that were different in these websites and that needed other approach and a separate implementation.

It is the `Spec` class the one that contains all the specific methods that will provide those steps that are different for the website we are extracting data from. Here we identified those steps that might be different, so they must be implemented by the developer who uses the provided framework. Those methods are:

- `getDatum();`
- `getURL();`
- `getTitle();`
- `getCurrency();`
- `getPrice();`
- `extraSteps();`

What this means is that all those methods above must be considered for implementation by the developer depending on the website that needs to be crawled. However, some of those methods might not be implemented, depending on the needs of the specific website, but anyway the developer must at least consider them and, if for any reason some of them are not needed, they can be skipped.

It is also important to consider that a method to specify sleeping times or breaks between the requests to the website has not been implemented. This is because HtmlUnit already provides a setting that automatically adjusts this timing in order to be able to obtain the dynamically updated page.
4.7 Problems

The solutions presented above have some problems and drawbacks that will be reflected here. When it comes to crawling and robots software, general approaches tend to be fast and efficient, finding and extracting data from websites at a relatively fast speed. One of the main problems that can be found in the solution proposed through this thesis is that its speed is much slower than the approaches for other more general sites. This is due to the complexity of the specific websites that this piece of software is focused on. Dealing with AJAX websites that load its content dynamically makes it compulsory to simulate the interaction with the website in order to crawl and extract all its information. In order to simulate this interaction, as it was mentioned before, HtmlUnit was chosen. This software simulates and do exactly the same requests as a generic web browser would do. This means that this solution needs to make exactly the same requests and consequently wait for the same responses as a web browser would receive, and this happens for each page of the tables that we are crawling. In order to show it clearly, MorningStar has an average loading time per page of 7.7 seconds, while for Fondmarknaden is only 2.5 seconds. In the following figure it is graphically shown this difference:

![Loading times graph](image)

Figure 4.13: Loading times of 3 requests of pages for MorningStar and Fondmarknaden.

Another problem that we found during the implementation of this solution comes also with the HtmlUnit tool. After some research, it is found that this tool has some memory leak problems sometimes. As they belong to an external tool used for this thesis, this is not considered as a main problem to solve the research questions of this project, but it becomes a problem in websites with a lot of information when the software has to crawl through many different pages. In general this problem was solved adding more heap memory to the JVM through the configuration, while running the Java programs, but it is still a problem worth to mention.

This memory leak is relevant for the implemented robots only within an iteration, as it depends on the number of pages where the website has the funds distributed. HtmlUnit opens what it is called a WebWindow for each page. HtmlUnit should close them once...
used, but some times that doesn’t occur. However, as mentioned this problem has been solved through the JVM configuration, and at the same time, in the implementation the WebClient is restarted in each iteration in order to avoid any possible problem. This means that, once all the funds have been obtained, no more problems regarding this memory leak should appear throughout the successive updates and iterations.

Regarding the selected websites in which the implementations were based, some problems could also be found. For this project it is important to describe and identify the information in a unique way. This information in the studied scenarios were funds, and the way to identify them changed through the implementation of the tool. In a first iteration, the name or title of the fund was considered as the unique identifier, checking afterwards the price of that fund in order to detect possibles updates. After testing the tool, it could be appreciated that in the studied websites more than one fund had the same name or title. Once seen that, this definition was changed and the currency of the fund was added, being this the final modification of the way the funds are identify. Nevertheless, this is still a minor problem in some cases, as there are some funds that share the same title and currency, but that have different price.

As it is shown on the figure below, which belongs to MorningStar website, we couldn’t find a unique way to identify or differentiate one fund from the other, as they share title and currency, and the price can not be considered as an identification part as it will change in future updates (and it is the main purpose of the tool, to detect these updates on prices of the funds).

Figure 4.14: Duplication of funds in MorningStar website.
In this project, we end up considering any fund that can’t be distinguished as an update of the previous fund with the same name and currency. This makes it to give false updates for those funds, but we could not find a better solution for this problem.
5 Results

The results of the implemented application consist on the success of running it and retrieving all the available economic data available on the mentioned websites and pages. In order to obtain those results, the application had to be running for over 24 hours, in order to detect any possible update on the data and also to confirm that this application can be running for a long period of time, as it is its purpose further on.

Due to the nature of this application as a robot, the obtained data can be a bit chaotic shown in plain text, so for this reason some of this data will be analysed and presented in graphs, trying to show all the possibilities and how far this application got.

As the application has crawled and extracted data from the presented websites, it is important to understand how much data those websites contain. Even when they are constantly changing as new funds might be added or updated, some numbers will be given:

![Figure 5.15: Graph showing the needed time to extract funds in MorningStar and in Fondmarknaden.](image)

Fig 5.15: Graph showing the needed time to extract funds in MorningStar and in Fondmarknaden.

First of all, MorningStar has around 21000 funds (21085 as 16th of May of 2016), presented as 20 funds per page in a total of 1055 pages. In order to crawl this information, the application run for around 3 hours (2.83 hours in the supervised run). Due to the amount of data a specific number of correct/incorrect data collected can not be provided.

Secondly, Fondmarknaden has around 1700 funds, presented as 30 funds per page in a total of 57 pages. The total running time needed by the application to obtain these data was 6:27 minutes in our supervised run.

Last, Avanza has around 1300 funds. As it can be seen in the graph, and as we explained before, this website will not be considered for the following results, as it is a
different kind of website where the obtained data will be of no relevance for this comparison. This is due to that this website is static and simpler to crawl, which means that, as it can be seen on the graph, it is much faster to handle and will not present all the problems that the other two websites present.

Both of the dynamically updated websites might have quite different results while crawling them depending on many factors, as for examples the errors that the application retrieves while trying to extract the data, the quality of the internet connection and the time that the website needs to answer our requests.

But as this project consisted in developing a framework, the implementation itself should be also considered as a result. The implementation of the robots for both MorningStar and Fondmarknaden have been implemented both using and not the mentioned framework. This framework does not involve Avanza due to the mentioned reasons. In order to consider this way of implementation as a result, many factors can be mentioned.

![Figure 5.16: Graph comparing the lines of code needed for the robots using or not the implemented Framework.](image)

In order to understand how this framework can help in developing a robot of these characteristics let’s compare the implementation that uses the framework and the one that doesn’t for both websites: Starting with the MorningStar robot, its implementation without framework is about 328 lines, while the implementation using the framework is only 72 lines. Comparing now Fondmarknaden, its non-framework implementation takes 357 lines, while the framework one is 83. In both implementations that used the framework, it was only needed to implement specific code that involves the structure of the website. For that it is only considered the code that extends the Spec class, which is considered the specific implementation for each website using the framework.

Finally, and considering the framework itself as a result, the code of this project will be publicly available in the following repository: github.com/xulegaspi/BachelorProject
6 Analysis

As it can be seen from the provided results, we have implemented an application that is able to crawl and automatically extract economical information from dynamically updated websites. The obtained results show a huge amount of data that has been automatically gathered, which means that it provides a service that can save a lot of time to a final user interested on this kind of economical data.

As it was mentioned in the introduction of this thesis, this project belongs as a part of the LnuDSC project. Taking this into consideration, this tool solves the problem presented in those websites that can not be crawled by more generic robots, the so called “dynamically updated” websites. Considering the complexity of this websites and how different they can be, this tool provides an approach on how to deal with such pages.

Furthermore, after generalizing and implementing the framework, and as it is shown in the results, the amount of code needed to crawl and extract information from one of these websites is considerably reduced. This should imply also a reduction of the time needed to implement a robot of these characteristics.

Now focusing on the results given related to the time needed for extracting the different funds in both MorningStar and Fondmarknaden, the differences that can be appreciated are probably derived from the different number of funds shown per page. As we have explained before, using HtmlUnit allow us to interact with the website, but this also has as drawbacks to expect the same problems as if we were the final user. This means that the main problem while crawling and extracting these data is the waiting time that this tool introduces. In more generic robots, the time needed to crawl and extract information is much lower than the one needed for this tool. This is due to the time needed by the server to retrieve each page of the tables we are requesting. As those pages are not reachable by a unique URL, we need to request them using their own JavaScript methods, and in the end this become a huge amount of time needed to crawl the whole dataset. It is because of these that the difference shown in the needed time per fund can be explained. In Fondmarknaden, 30 funds are shown per page, while in MorningStar only 20 can be found. This means that in Fondmarknaden we can extract more funds per page. Extracting the funds in a single page is not time consuming if we compare it to the time needed to retrieve a page itself. That is why in MorningStar, where it is needed to request more pages for the same number of funds, the final time-per-fund needed is higher. Considering a mathematical perspective, the ratio expected between the times-per-fund of both websites should be 2/3, as in MorningStar for crawling 60 funds we would need to request 3 pages, but for Fondmarknaden we only would need 2. However, the ratio obtained is not the mathematical one. This can be for many reasons, as for example different response times for the different websites or a different load on the machine running this tool.
7 Discussion

As it has been shown in this thesis, the implemented robot and framework provides a tool that can crawl and extract data from dynamically updated websites with economical data. This means that the main research questions presented have been answered, and also their benefits and drawbacks have been presented.

Also referring to the previous research, we can verify how the selected tool, HtmlUnit, have been also used for other researches trying to provide a solution with similar problem [18] [1]. This makes clear that so far some of the drawbacks mentioned for this tool haven’t been solved yet, and also other researches had dealt with the same problem.

There are also further studies where the solutions are much more general, as the dynamically updated website is analysed by the own application [19], but this results also in a much more complex application that goes over the magnitude of this project, adding unnecessary complexity for the purpose we wanted.

The data obtained by the implemented application through this thesis shows how it is possible to extract information from the specific websites mentioned. From here we understand that other similar websites are also possible to crawl. Also, by implementing the presented framework, we believe that further implementations using this framework should be a much faster process than it was for this project, helping developers to implement more robots that can collaborate into the LnuDSC project.
8 Summary, Conclusion and Future Work

After all the work carried with this project, we can now present a useful framework that explores the development of robots for specific dynamic websites that haven’t been studied or implemented before. A tool like this has endless possibilities, just as endless dynamic websites can be found nowadays on internet (and those that are being created constantly).

As presented on this project, it is possible to implement specific robots to some websites where general robots can not extract the desired data. We have studied and explained how to implement a robot, and using the implemented framework in order to help future developers, that can scrap economical data from AJAX or dynamically updated websites where the interaction of the user seems needed.

It is now the time to use this tools and this knowledge in order to find possible errors that couldn’t been found through the course of this thesis. There are several websites where it can be tested, and where as a future work would be very interesting.

We hope that with the results and tools provided with this thesis we can improve and understand a bit better how the web is evolving and how dynamic websites work, being at the same time able to extract some information from them. Because this might be a specific area of the web that haven’t been as studied as other more generic areas, this thesis has addressed some still unresolved problems.

This research is relevant for science as it complements other researches on the area. It can also be relevant to provide a specific solution for the robots of the LnuDSC project.

As an important and pending future work for this thesis, other economic dynamic websites should be tested. As it doesn’t have to be in any specific language (this thesis has been carried out with Swedish economic websites even when my Swedish is quite limited), here it is addressed an Spanish website that might be eligible for a future testing: bbva.com/funds [20].

Testing and developing a robot using the given framework for other websites would completely prove the usefulness of the framework and of the tools.

8.1 Future Research

The chosen project and how it was conducted was in a very specific direction due to the scope and time limitations. However, there is much more that could be done.

First of all, given the implemented tool, a further and much more exhaust evaluation could be made. More testing could provide some errors that couldn’t be found through the followed methodology, and as it is known in software technology, a developer can never be sure that the implemented software has no errors, the only option is to test it as much as possible to avoid most of the errors.

Also in order to evaluate this tool, it would be very interesting to test with other developers to implement a robot using the given framework with a controlled environment,
given a specific website. Then comparing the time spent by those developers implementing this robot using the provided framework and without using it, and later on making a survey that could retrieve some insights about this tool. That would help with some feedback about the framework that would probably help to improve it in many different ways.

Another possible next step that will be needed is implementing and combine this tool with the whole LnuDSC system. That would also need to solve some compatibility problems probably, even when this thesis has been done under the LnuDSC context, which means that the communication protocol followed (sending the data as JSON) has been already adopted in order to avoid future problems.

Finally, as mentioned before, this tool has been developed for a very specific scenario. However, this scenario could be easily made wider, as there are many dynamically updated websites that don’t provide economical data but that have a similar “table-based” structure. Also as mentioned in the previous research, there are some studies about tools that automatically analyses the website’s structure, providing a wider range of websites that can be crawled by the tool. Combining this studies with the provided framework would be a really interesting future work that could solve many problems while extracting data from dynamically updated websites.
References


