DESIGN AND IMPLEMENTATION OF AN AUTOMATED WEB SCRAPING SYSTEM: ENHANCING ACCURACY AND EFFICIENCY FOR NETTILEASING FINLAND OY

Lappeenranta–Lahti University of Technology LUT

Master's Programme in Software Engineering and Digital Transformation

Master's thesis

2023

Arsalan Khan

Examiners: Associate Professor Annika Wolff

Junior Researcher Ajesh Kumar

Abstract

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Master's Programme in Software Engineering and Digital Transformation

Arsalan Khan

Design and Implementation of an Automated Web Scraping System: Enhancing Accuracy and Efficiency for Nettileasing Finland Oy

Master's thesis

2023

79 pages, 19 figures, 8 tables

Examiners: Associate Professor Annika Wolff and Junior Researcher Ajesh Kumar

Keywords: Web Scraping, E-commerce Data Collection, Operational Efficiency, Scrapy and Playwright, Dynamic Content Extraction

The objective of this thesis to address specific research questions through the development of an automated web scraping tool for Nettileasing Finland Oy, a startup based in Lahti. This study explores the implications of web scraping in a business context, evaluating its efficacy and potential in streamlining data acquisition processes. It attempts to enhance the efficiency and accuracy of data gathering from e-commerce platforms. The study focuses on three websites: bikemarine.fi, venekauppa.com, and tietokonekauppa.fi. It uses tools like Python, Scrapy, and Playwright for scraping product details. These tools are capable of handling both static and dynamic web content.

The study covers the design, testing, and implementation of the system. It highlights the system's ability to adapt to different web structures. It also demonstrates efficient data extraction. The study evaluates the system's performance against manual methods showing significant improvements in data accuracy and speed.

The thesis also considers how the system affects business operations. It focuses on improved insights and possible revenue growth for Nettileasing. The study concludes by emphasizing the importance of automated web scraping in e-commerce. It recommends using AI and machine learning for more in-depth data analysis in the furture.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Annika Wolff, for her invaluable guidance, patience, and expert advice throughout the course of this research. Her profound knowledge, attention to detail, and support have been instrumental in shaping this thesis. Her encouragement and insightful critiques have inspired me to push my boundaries and achieve a higher standard of work.

I am also profoundly grateful to Nettileasing Finland Oy for their collaboration and support in this project. The opportunity to work on a real-world application and the insights provided by their team have been pivotal in the practical understanding and implementation of the concepts studied. Their willingness to share their expertise and resources has greatly enriched my learning experience and contributed significantly to the success of this project.

A special word of thanks goes to my mother, whose love, support, and sacrifice have been the bedrock of my life. Her endless encouragement and belief in my abilities have been a constant source of strength and motivation. This thesis is not only a reflection of my work but also a testament to her unwavering faith in me.

To all those who have assisted me, offered insights, and encouraged me throughout my academic journey, I extend my heartfelt appreciation. This achievement is not solely my own, but a culmination of the support and guidance of many.

Table of contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

The emergence of the internet era has brought a dramatic change in how consumers and corporations conduct business. E-commerce, which was once an emerging industry has grown into a global juggernaut and changed the way retail is streamlined. Statista figures show that the European E-commerce business is expanding and might reach 900 billion USD by 2027 (Ecdb, 2023). The shopping experience has also become more accessible. However, it has also added a new dimension to the customer experience by recognizing convenience, choice, and satisfaction. E-commerce platforms have crossed geographic boundaries. From booming metropolises to isolated villages, it brings a variety of goods to consumer's doorstep.

Although the digital revolution has increased the range of options, the growing number of online shopping possibilities has several drawbacks. The abundance of options accessible to customers makes them happy, but the sheer amount of information also overwhelms them. The cognitive stress and decision paralysis that customers frequently endure in the face of a wide range of options is explained by the paradox of choice, as described by psychologist Barry Schwartz (Schwartz, 2004). This translates to the challenging task of sorting through a multitude of products, deals, and retail stores in the world of online shopping. The challenge of navigating through this maze sometimes leaves customers feeling overwhelmed and irritated. This is where the importance of simplicity and ease of navigation becomes critical. The burden of making decisions can be greatly reduced by a platform that streamlines the process by carefully curating options and streamlining comparisons (Shaikh et al., 2023). Such platforms encourage customer retention while also enhancing the shopping experience. In a crowded market where distinction is critical, facilitating consumer decision-making can be a game-changer for e-commerce platforms, particularly ones focused on price comparisons.

Consumers have become more and more reliant on price comparison websites to sort through this complicated web of choices. These platforms gather product prices from numerous online shops and display them to customers in a streamlined and condensed way through their complex algorithms and enormous databases. These platforms empower customers by providing a comparative overview, encouraging educated choice-making and ensuring they get the most value for their money. These price comparison websites are built around the intricate web scraping approach. In this digital age, this method of programmatically pulling data from websites has become essential. Web scraping tools crawl the internet, gathering enormous amounts of data that are fed into the databases of price comparison platforms (Diouf et al., 2019). These tools range from simple scripts to sophisticated bots. In stark

contrast to manual approaches, this automated data collection method promises efficiency, scale, and accuracy.

Web scraping, a technique that dates back to the early days of the internet, has progressed dramatically over the last few decades. Web scraping was initially a crude process that frequently involved data extraction from static web pages. But as dynamic websites emerged and web content became complex, it became clear that advanced, automated web scraping techniques were necessary. Today, web scraping is a highly developed discipline that combines elements of many other disciplines, such as data science, machine learning, and artificial intelligence. It has become a driving force for directing research, advancing innovation, and establishing corporate strategy.

Despite the advancements and capabilities of web scraping tools, they are not without challenges. Traditional scraping solutions often struggle with handling dynamic content, such as AJAX calls and dynamically loaded web elements. Additionally, websites are increasingly deploying anti-scraping mechanisms, including CAPTCHAs and user-agent string checks. Maintaining web scraping scripts also presents a significant challenge, as the ever-evolving nature of web design necessitates frequent updates. Furthermore, web scraping operates in a complex legal and ethical landscape, often navigating a fine line between data accessibility and privacy concerns, which can give rise to legal issues.

1.2 Problem Statement

In the current digital era, e-commerce platforms, particularly price comparison websites are essential for assisting consumers in making purchases. The value proposition of these platforms are based on their ability to provide accurate and current pricing information. Nettileasing Finland Oy, with its ambition of becoming a trusted source for comparing monthly subscription and leasing products, stands at cross-roads of technical innovation and consumer expectations.

1.2.1 Nettileasing Finland Oy: A Case in Point

Nettileasing Finland Oy, a startup business in Lahti, specializes in providing a price comparison platform for products available on monthly subscription or leasing basis. Although the company already has a robust website infrastructure, The absence of an automated system for data collection is a big problem for the company. This gap significantly hampers the website's utility, as the core functionality of a price comparison site relies on the accuracy and timeliness of product data. The current situation leaves Nettileasing Finland Oy's website at a competitive disadvantage, as users expect real-time and accurate data when engaging with a price comparison platform. Furthermore, the absence of automation in data gathering

and implementation imposes an unsustainable stress on employees who must participate in burdensome and endless manual data collection and implementation.

There are several challenges that are currently faced by Nettileasing oy:

- ***Manual Data Collection:*** Nettileasing currently collects data manually, which is labor-intensive, time-consuming, and prone to mistakes. The accuracy of the data supplied to users is compromised by this manual method. This jeopardizes the platform's reliability, potentially jeopardizing user trust and satisfaction.
- ***Lack of Real-time Updates:*** Due to the lack of an automated system, the platform struggles to give real-time price updates, a feature that has become virtually a standard expectation for modern consumers. Furthermore, the lack of real-time data updates might lead to missed opportunities and consumers making incorrect decisions.
- ***Scalability Issues:*** Manual data collecting becomes increasingly unsustainable as the platform expands. Manual updates are a growth hindrance for the platform because of the enormous amount of data points needed, from specific price details to product images.

The challenge for Nettileasing Finland Oy is not simply to use scraping tools and technologies, but to optimize them to guarantee the highest levels of efficiency and accuracy in data collection. This optimization is essential, especially in context of the volatility of online pricing and the enormous volume of data that must be gathered and updated often. Therefore, the fundamental research challenge is how to build and execute an automated web scraping system to satisfy these requirements while maintaining the platform as a reliable information source for its users.

While conventional web scraping technologies serve as a starting point, Nettileasing's unique challenges demand a more tailored approach:

- ***Diverse Data Requirements:*** The data landscape is large and diverse, ranging from product images and descriptions to specific price details.
- ***Uncompromising Accuracy:*** Data accuracy is essential in the realm of price comparison since even small errors can have a big impact.
- ***Seamless Integration:*** Real-time data availability is ensured via seamless platform integration, which goes beyond simple data extraction.
- ***Scalability:*** As Nettileasing broadens its scope, the scraping system must change in order to handle more websites and more data effectively.

1.3 Objectives of the Study

Nettileasing Finland Oy has obtained data scraping agreements with three websites: tietokonekauppa.fi, venekauppa.com/fi, and bikemarine.fi. A working web scraping software with a user interface is required as a Minimum Viable Product (MVP). This application should be able to process the relevant data from these websites and save it in a file (CSV, JSON) for later input into the Nettileasing website. The essential data to be collected includes product links, images, brand, model, descriptions, monthly and leasing prices, location, and number of payments. The overall aim of this thesis is to improve the effectiveness and efficiency of data gathering in the e-commerce industry, specifically for Nettileasing Finland Oy. It is motivated by a number of specific objectives.

1.3.1 Technical Objective

Development of a Web Scraping System: The main objective is to develop an automated web scraping system specifically for Nettileasing Finland Oy. This system is designed to address the company's immediate need for accurate and timely data collection from various e-commerce websites.

System Specifications: The system aims to automate the process of gathering product information which including prices, descriptions, images, and other relevant details, from predetermined websites. This automation is expected to replace the current manual methods, thereby streamlining the data collection process.

Technical Requirements: The development of this system involves addressing technical challenges such as handling dynamic content, managing data extraction from multiple sources, and ensuring the scalability and reliability of the system. The goal is to develop a robust, user-friendly solution that can be smoothly incorporated into Nettileasing Finland Oy's existing website.

1.3.2 Academic Objective

Efficiency and Accuracy: One of the key objectives of this research is to assess the effectiveness and accuracy of the built web scraping system. This entails comparing the system's performance against traditional manual data gathering techniques, with a focus on criteria such as data collection speed, data extraction accuracy, and the system's capability to update data in real-time.

Customer Satisfaction and Retention: Another important goal is to determine the possible impact of improved data collection methods on customer satisfaction and retention. This

assessment will look at whether the more accurate and timely data given by the web scraping system results in a better user experience on the Nettileasing Finland Oy website, which might increase customer loyalty and engagement.

Practical Implications: The goal of the study is to give insight into the practical implications of implementing an automated web scraping system in an e-commerce setting. It aims to provide evidence-based findings on how such systems might maximize data-driven e-commerce strategies, eventually benefiting both the business and its consumers.

By achieving these goals, the study hopes to make a significant contribution to the field of e-commerce by providing a practical solution to a common challenge faced by many businesses in the sector, while also advancing theoretical understanding of web scraping technologies and their applications.

1.4 Research Questions

The challenges faced by Nettileasing Finland Oy give rise to two pivotal research questions that this thesis aims to address:

1. How does the implementation of an automated web scraping system improve the accuracy and efficiency of e-commerce products data collection compared to manual methods for Nettileasing Finland Oy?

    - This question investigates the potential enhancements in data collection processes through automation. It compares the effectiveness of the online scraping technology in terms of data accuracy and operational efficiency to traditional manual data collecting techniques.

2. Does the improving accuracy and efficiency of data collection lead to improved customer satisfaction and retention?

    - This question delves into the broader impact of the web scraping system on the end-user experience. It investigates whether improvements in data gathering accuracy and efficiency translate into improved customer happiness and retention, both of which are essential success indicators for any e-commerce platform.

Addressing these research questions will not only provide Nettileasing Finland Oy with a viable solution to its current challenges but will also contribute to the broader understanding of the role of web scraping in enhancing e-commerce operations. The findings from this study are expected to offer valuable insights into the optimization of data-driven strategies in the e-commerce sector, ultimately benefiting both businesses and consumers.

1.5 Thesis Structure

The structure of this thesis is designed to systematically guide the reader through the various stages of the research, from the initial framing of the problem to the final conclusions and recommendations. The thesis is organized into six main chapters, as follows:

Chapter 1: Introduction

This initial chapter sets the stage for the research by providing a comprehensive background on the evolution of e-commerce and its current landscape. It delineates the problem statement, focusing on the specific challenges Nettileasing Finland Oy faces in data collection and the need for an automated solution.

Chapter 2: Literature Review

In this chapter, a thorough review of existing literature is presented, covering key themes such as the development of web scraping technologies, their application in e-commerce, and the associated legal and ethical considerations. It also explores the challenges and solutions in web scraping, including aspects of scalability and community support, setting a theoretical foundation for the research.

Chapter 3: Methodology

This chapter describes the research methodology adopted for the study, focusing on the 'Research Through Design' approach. It details the data collection techniques and analysis methods used.

Chapter 4: System Design and Implementation

Here, the focus shifts to the practical aspect of the thesis, the design and implementation of the web scraping system for Nettileasing Finland Oy. This chapter details the system architecture, the integration of key technologies such as Scrapy and Playwright, the data collection and processing mechanisms, and addresses the handling of dynamic content and other technical challenges encountered during development.

Chapter 5: Results

This chapter evaluates the performance of the developed web scraping system, comparing it against manual data collection methods to assess its efficiency and accuracy. It also examines the impact of the system on business operations and customer retention at Nettileasing

Finland Oy, providing valuable insights.

Chapter 6: Discussion

The discussion section of a thesis is important for integrating research findings and connecting them to the larger context of current literature and research questions.

Chapter 7: Conclusion and Recommendations

The concluding chapter summarizes the key findings of the research and draws conclusions based on the study's objectives and research questions. It offers recommendations for future research in the field and discusses the practical implications of the study's findings for the e-commerce sector, particularly in the context of web scraping technologies.

2 Literature Review

Although this review has a broad scope, it is important to define its boundaries. We will follow the evolution of e-commerce, seeing its change and the increasing role of data in defining consumer experiences and corporate strategy. This background information lays the groundwork for a greater understanding of the tools and methods that have been developed. Then we will look into the technical details of web applications and how they work. After that, we will look closely at the technical nuances of web scraping as well as its methodology, tools, and applications. The discussion then moves to the technical side, breaking down the many tools and technologies that are essential to web scraping. This section attempts to give a thorough overview of the current technologies, emphasizing the ways in which these tools have developed and are applied across a range of industries. We then move to a thorough examination of the legal and ethical ramifications of web scraping, to assist readers in navigating the complex web scraping landscape, understanding the delicate balancing act between ethical obligation and technical innovation. After that we discuss the difficulties faced by web scraping techniques, creative solutions and best practices used by specialists in the field to get beyond these obstacles. We then investigate the scalability and growth of web scraping projects. This section summarizes the collaborative and dynamic nature of web scraping in the contemporary data ecosystem by highlighting the contributions made by the community in progressing the field.

To sum up, the goal of this literature review is to present a thorough, comprehensive analysis of web scraping and data gathering within the context of e-commerce, laying a solid groundwork for the remaining chapters in this thesis.

2.1 The Evolution of E-commerce and Data Collection

E-commerce has grown from its infancy in the early 1990s to become a dominating type of modern retail (Turban et al., n.d.). Technological developments, greater usage of the web, and the rapid growth of mobile devices have accelerated this progress. Data gathering and analysis have been critical factors in this expansion, enabling targeted marketing, efficient supply chain management, and improved consumer experiences (Chaffey & Ellis-Chadwick, 2019). As e-commerce platforms become more complicated and data-driven, there is a greater demand for sophisticated web scraping solutions that can handle a variety of data sources and formats. These solutions are necessary for gathering various product data from numerous websites, which is an important aspect in the business model of companies such as Nettileasing Oy in Finland.

In the early days of e-commerce, organizations largely collected data through consumer

surveys and sales records. However, these solutions were sometimes time-consuming and limited in scope, making it difficult to quickly adjust to market changes and customer needs. With the introduction of Web 2.0, businesses began to leverage user-generated content and behavioural data, which enabled more in-depth understanding into consumer preferences (Kozinets, 2002). This marked the emergence of "big data" in e-commerce, a term used to describe the enormous volumes of both structured and unstructured data that companies gather, usually in real-time. For a company like Nettileasing Oy, which focuses on collecting product details from various websites for their users, this shift highlights the importance of developing a web scraping system that not only efficiently harvests vast quantities of structured and unstructured product data but also processes and analyzes it for accurate, real-time market and product insights.

The advancement of machine learning and artificial intelligence has increased the importance of data analytics in e-commerce. Based on the data collected, algorithms can now forecast consumer behavior, automate customer service, and improve inventory management, resulting in a more dynamic and responsive e-commerce model (Jordan & T. M. Mitchell, 2015, Rust & Huang, 2014). However, this reliance on data collection has led to concerns about data security and privacy, prompting legislation such as Europe's General Data Protection Regulation (GDPR) (Voigt & Bussche, 2017, Intersoft, 2022). This is particularly relevant for our project, as we must navigate these regulations while collecting product data from various online sources.

The integration of data gathering technology with e-commerce platforms has become critical for effective business strategies. Data-driven decisions can now be made by even small-scale e-commerce platforms using tools like google analytics and customer relationship management (CRM) systems (Peelen & Beltman, 2013). The capacity of e-commerce businesses to efficiently obtain, analyze, and use data is becoming increasingly important (Chen, Preston & Swink, 2015).

When compared to previous data collecting methods, modern online scraping methods provide significant improvements in accuracy and efficiency. One interesting case study that demonstrates the impact of this comes from the wine industry. In (Jorge et al., 2020), an application was developed by "QuieroVinos, S.L.," an online wine shop, to monitor competitor prices and modify product prices competitively in different marketplaces. Sales and transportation expenses were significantly improved as a result of this strategy. Due to the frequent market price fluctuations, the application allowed real-time pricing comparisons, which would have otherwise been a time-consuming task. This case study shows how competitive pricing and quick response to market changes may be improved by automated data collection using web scraping and improve customer happiness and business success.

For businesses, the integration of sophisticated data collection tools with e-commerce platforms has become crucial. These tools allow even small e-commerce businesses, such as Nettileasing Oy, to make data-driven decisions. Web scraping in particular, promises significant gains in data gathering accuracy and efficiency over older approaches. Web scraping offers a complete and real-time insights of market trends and product availability by automating the process of obtaining precise product information from a large number of websites. The ability to automate the collection of product details from various websites and show these offerings in real-time can significantly impact customer satisfaction and retention. The case of QuieroVinos, S.L (Jorge et al., 2020). highlights the potential benefits that Nettileasing Oy could realize through the implementation of advanced web scraping techniques. This level of data richness is vital for Nettileasing Oy, enabling them to quickly respond to market changes and cater to customer preferences with an updated and diverse product portfolio on their platform. The implementation of web scraping represents a significant leap in data collection, aligning with the ever-increasing complexities of the e-commerce landscape and directly influencing customer satisfaction and retention through improved user experience.

2.2 Introduction to Web Applications

2.2.1 Web Application Components

Understanding Web application architecture, which is divided into client-side and server-side components, is critical for the design and implementation of web scraping strategies. The client-side is the primary point of interaction for web scraping technologies, mostly involving the user interface developed with HTML, CSS, and JavaScript. Understanding the complexities of these front-end technologies is critical for our project at Nettileasing Finland Oy since it defines how we extract data. For example, the dynamic nature of JavaScript-generated content presents unique challenges, necessitating more advanced scraping tools capable of rendering and interacting with JavaScript-based content.

Client-Side Components

A web application's client-side environment is what the user interacts with. It is mostly comprised of the user interface and is developed with HTML, CSS, and JavaScript. The following are the front-end components:

- ***HTML (Hypertext Markup Language):*** This serves as the foundation for all web pages and applications, organizing and creating sections, paragraphs, headers, links, and blockquotes. Scraping HTML is simple but restricted in scope since it does not account for dynamically loaded content. This constraint is crucial when dealing with current e-commerce platforms, which often display product listings and pricing using

dynamic content.

- ***CSS (Cascading Style Sheets):*** The webpage's design, or how the site feels and appears, is made using CSS. It enables web pages to adjust to the dimensions of various devices, including phones, tablets, and desktop computers. Although web scraping does not directly target CSS, it is necessary to understand its structure in order to locate and extract data with accuracy. CSS selectors are used to enhance the precision of web scraping by identifying specific data points inside a web page.
- ***JavaScript:*** JavaScript adds functionality to web pages for specific purposes. It may be used to create cookies, recognize the user's browser, and provide interactive elements such as slideshows, forms, and so on. This presents an additional difficulty for online scraping, as typical HTML parsing tools are insufficient. Our approach requires the use of tools like Playwright and Scrapy, which can render and interact with JavaScript, to efficiently scrape content generated by JavaScript. This method is critical for correctly collecting the dynamic and interactive contents that is used by e-commerce websites.
- ***Frameworks and Libraries:*** Developing parts of client-side code can be made simpler by using one of the many JavaScript frameworks and libraries available, such as React.js, Angular.js, and Vue.js, which offer pre-written JavaScript code for common programming functions and tasks.

Server-Side Components

A web application's server-side (or back-end) is where data is stored and maintained. It is the component of the application that we do not see but that is critical to its operation. The following are common back-end components:

- ***Server:*** This is the machine on which the web application is installed. It receives client-side requests, processes them, and returns the relevant data responses. To efficiently obtain data, our scraping techniques must account for various types of databases and server responses.
- ***Database:*** Databases organize and store data so that web applications may search and retrieve it as needed. Commonly used databases include NoSQL databases like MongoDB and SQL databases like PostgreSQL and MySQL.
- ***Server-Side Languages:*** These are the programming languages used to manage the server's logic. They collaborate with the database and server to process the application's business logic. PHP, Python, Ruby, and server-side JavaScript (Node.js) are examples of popular server-side languages. For our web scraping strategy to be effective, it has to be flexible enough to accommodate different server-side technologies.

- ***Web Server Software:*** Software like Microsoft's Internet Information Services (IIS), Nginx, and Apache process HTTP requests from clients and decide how to respond. They can return a 404 error, call a server-side language, or deliver a static file. The availability of data for scraping is influenced by the behavior of web server software, particularly in the way it handles HTTP requests. Identifying server responses like 404 errors is essential for creating effective scraping solutions.

2.2.2 Web Application Rendering Process

The process by which web applications show the requested pages to the user is known as rendering. There are numerous ways to present web application content:

- ***Server-Side Rendering (SSR):*** When a new page is requested by the user, the HTML is created on the server. It is widely used to show static website content and web pages with information that does not update regularly.
- ***Client-Side Rendering (CSR):*** CSR provides the browser with a basic HTML page together with a JavaScript file that the browser needs to display the page's content. In single-page applications (SPAs), where web application interaction is more fluid and dynamic, this technique is commonly used.
- ***Hybrid Rendering:*** Both approaches are regularly used in modern web-based applications. The server handles the first rendering, while CSR handles all further interactions. This blends dynamic interaction with quicker load times, the best of both worlds. This hybrid approach requires our web scraping technologies to be capable of handling it in order to guarantee accurate and rapid data extraction.

The effects of rendering processes on web scraping are not independent. For example, server-side rendering makes it simpler to scrape data since it updates automatically without requiring a new request to the server. On the other hand, in order for client-side rendered apps to access dynamically loaded content, more advanced scrapers are needed, such as those that can run JavaScript and simulate user interactions.

When developing a web scraper, it is essential to be aware of these aspects and methods as they determine the technologies to be used and the structure of the scraper, which in turn determines how well the data is retrieved. This information not only facilitates effective data retrieval but also guarantees the accuracy and completeness of the data collected, both of which are critical for making well-informed decisions in the e-commerce industry.

2.3 Web Scraping and Web Crawling

Web scraping and web crawling are crucial methods for gathering data from the internet. They provide a foundation for several applications across various industries, such as search engines, e-commerce, and data mining (R. Mitchell, 2018). Web scraping and web crawling are used interchangeably, however they have different goals and use different techniques.

Web Crawling

This is often the initial stage and refers to the act of methodically searching the Internet in order to index and find the latest and most updated contents. Web crawlers, sometimes referred to as spiders or bots, begin with a seed list—a collection of URLs to visit. Recursively traversing the web in accordance with a set of policies, the crawler examines these URLs, finds every hyperlink on the page, and adds it to the list of URLs to visit next (Olston & Najork, 2010).

Two key features of web crawling are the selection policy, which controls the order in which URLs are visited and makes sure the most essential content gets examined first, and the politeness policy of the crawler, which prevents web servers from being overloaded with requests (Heydon & Najork, 1999). One of the most well-known applications of large-scale web crawling is Google's search engine, which updates its index regularly to reflect changes on the web (Brin & Page, 1998).

Web Scraping

It is a method for obtaining certain information from websites. A web scraper is an application that opens a web browser, grabs webpages, extracts the necessary data, and then transfers that data into an XML file, database, or spreadsheet (Munzert et al., 2014). Rendering the page, parsing the HTML to find the data, and finally gathering the data are the usual steps in the process. This is especially important for e-commerce businesses that depend on current product descriptions, prices, and availability information from competing businesses or suppliers.

A wide range of technologies are used in web scraping, from basic HTTP queries to get static page content to sophisticated headless browsers capable of executing JavaScript and managing dynamic AJAX calls. When working with JavaScript-heavy websites, tools like Selenium and Puppeteer are utilized, whereas Python programs like BeautifulSoup and Scrapy are popular choices for extracting static contents (Richardson, 2023).

Even though web scraping is useful, there are technical hurdles to overcome. These include

maintaining sessions and logins, handling pagination, navigating various website layouts, and getting past anti-scraping methods implemented by certain websites. Furthermore, as not all data is available for scraping and since scraping operations may violate website terms of service and data protection legislation, ethical and legal issues must also be taken into consideration (Krotov & Silva, 2018).

Web scraping is used in e-commerce not just for pricing strategies but also for tracking consumer activity patterns, monitoring customer feedback and opinion and improving search engine visibility through SEO (Mithas, Ramasubbu & Sambamurthy, 2011).

2.4 Web Scraping Technologies and Tools

Web scraping technologies have advanced significantly, and a wide range of solutions for data extraction from webpages are now accessible. This expansion is being driven by the increasing need for online data in many industries (Ismail et al., 2015). This section focuses on specific web scraping technologies and their use in e-commerce, with an emphasis on accuracy and efficiency.

2.4.1 Python-Based Web Scraping Tools

BeautifulSoup: A Python package that provides easy methods for navigating, searching, and modifying the parse tree, making it excellent for beginners and applications requiring speedy data extraction from static web pages (Richardson, 2023).

Scrapy: A collaborative framework that specializes in large-scale web scraping and crawling, allowing developers to effectively extract data from websites and expand the tool's capabilities (Kouzis-Loukas, 2016).

Requests-HTML: Combines a rendering environment based on Chromium with the parsing power of Python tools for scraping dynamic content (R. Mitchell, 2018).

LXML: Well known for its speed and adherence to web standards, as well as its sophisticated XML and HTML parsing capabilities (*lxml - Processing XML and HTML with Python* 2022).

2.4.2 Browser Automation Tools for Web Scraping

Playwright and Selenium: Automate web browsers to scrape dynamic content and simulate human interactions with web pages, but they have a steeper learning curve and use more processing resources (Lekh, 2023).

Puppeteer: Designed for Node.js, provides a high-level API over the Chrome DevTools

Protocol for rendering dynamic content and automating browser tasks (Google, 2023).

2.4.3 Alternative Web Scraping Libraries

Cheerio: A lean, server-side approach that suits individuals who are familiar with jQuery syntax, allowing for rapid HTML parsing without browser overhead (Kim, 2018).

Mechanize: The Mechanize package automates web interactions for scraping, however it lacks JavaScript processing, limiting its applicability in current web environments (Goyal, 2023).

jsdom: Mimics browser functionality in a Node.js environment, allowing JavaScript execution and DOM manipulation for scraping (APIFY, 2023).

Nokogiri: A Ruby module that allows for quick and easy HTML and XML parsing (Dalessio, 2023).

2.4.4 Applications of Web Scraping Tools

In (Shaikh et al., 2023), detailed a pricing comparison website that retrieved product information from several e-commerce websites using web scraping techniques. This website, which was created with PHP libraries for HTML parsing, demonstrates how successful web scraping is at finding customers the best deals while saving them time and money.

(Divya Khairkar, 2023), is another application which involves a real-time price tracker that checks product prices across many e-commerce websites. This application, which makes use of Selenium, Beautiful Soup, and the Requests library, demonstrates how web scraping may help consumers make educated purchase decisions by providing real-time price drop alerts.

In (Horch, Kett & Weisbecker, 2015), an innovative technique was created for automated identification and extraction of product pricing data from e-commerce websites, regardless of language or product domain. This technique is part of a study on data mining that shows how flexible web scraping is for gathering detailed product information from various e-commerce sites.

These studies demonstrate the practical value of web scraping in e-commerce, showing how various methods may be used to improve data collection accuracy and efficiency. Web scraping technologies play an important role in enabling consumers and companies in the digital marketplace, from price comparison to real-time tracking and extraction of specific product information.

Tools/ Frameworks	**Language**	**Dynamic Content Handling**	**Learning Curve**	**Speed**	**Browser Automation**	**Community Support**
BeautifulSoup	Python	No	Low	Medium	No	High
Scrapy	Python	No (extendable)	Medium	High	No	High
Playwright	Python/ JavaScript	Yes	Medium	Medium	Yes	High
Selenium	Python/ JavaScript	Yes	Medium	Medium	Yes	High
Puppeteer	JavaScript	Yes	Medium	High	Yes	High
Cheerio	JavaScript	No	Low	High	No	Medium
Requests/ Requests-HTML	Python	Partial (with Requests-HTML)	Low	Medium	No (Requests-HTML for JavaScript rendering)	High
HTTPRequest	JavaScript	No	Low	High	No	Medium
Beautiful Soup + Selenium	Python	Yes	Medium	Medium	Yes	High
LXML	Python	No	Medium	High	No	High
Mechanize	Python	No	Low	Medium	No	Medium
jsdom	JavaScript	Yes	Medium	Medium	No	Medium
Nokogiri	Ruby	No	Low	High	No	High
PhantomJS (deprecated)	JavaScript	Yes	Medium	Low	Yes	

Table 1: Comparison of Web Scraping Tools and Frameworks

Each of these frameworks and tools shown in Table 1, has special characteristics and ideal use. BeautifulSoup is a good option for novices and basic scraping tasks and it is frequently used together with Requests. Large-scale web scraping applications are better suited for Scrapy since it is more reliable. Tools that mimic an actual browser environment, such as Playwright, Selenium, and Puppeteer, are recommended for dynamic content, although they might not be as quick as headless tools and they have a steeper learning curve. Requests-HTML is a Python package that extends the capabilities of Requests to handle JavaScript by incorporating parsing features similar to those provided by BeautifulSoup, as well as a JavaScript environment for rendering. When speed is of the essence, LXML is frequently used since it parses XML and HTML documents very quickly and effectively. Mechanize, which is exclusively available in Python, imitate a browser to automate web interactions, although it does not support JavaScript. jsdom is a Node.js-based program that replicates

the browser environment and allows editing of web page structure and content. Nokogiri is a Ruby library for parsing HTML and XML that is well-known for its speed and simplicity. PhantomJS was previously a prominent tool for headless web browser scripting, but it has since been replaced by tools such as Puppeteer.

2.5 Legal and Ethical Considerations for Web Scraping in Finland

Although web scraping is an effective method for researchers and organizations to gather data from the internet, it is debatable if it violates ethical or legal norms. In Finland, like in many other places, the legality of online scraping is controlled by a combination of legislation, case law, and industry best practices. Below we will examine the ethical and legal framework that govern web scraping in Finland.

2.5.1 Legislations Affecting Web Scraping

Copyright Law: The Copyright Act (Tekijänoikeuslaki 404/1961), governs copyright law in Finland, which protects the expression of ideas but not the ideas themselves. Copyright may be violated by web scraping if it copies and utilizes copyrighted content without authorization (FINLEX, n.d.,FINLEX, 2023). Scrapers must use caution while copying protected content such as text, photos, and code without permission.

Database Directive: The EU Database Directive (96/9/EC), as incorporated into Finnish law, protects the significant investment made in developing a database. Extracting or reusing elements of a database without the permission of the owner may be illegal (EuropeanParliament, n.d.).

Computer Misuse Act: Legislation in Finland that is comparable to the Computer Misuse Act penalize unlawful access to computer systems, which includes breaching security measures and violating access rules while scraping the internet (Ministry-of-Interior-Finland, n.d.).

Personal Data Protection: The European Union's General Data Protection Regulation (GDPR), which also applies in Finland, limits the collection and use of personal data without explicit authorization (Intersoft, 2022). When scraping webpages, it is critical that we ensure that no personal data is gathered in violation of GDPR regulations.

Legal Article	Description	Implications for Web Scraping
Copyright Act (404/1961)	Protects the expression of creative works, not ideas.	Web scrapers must not copy copyrighted material without permission.
EU Database Directive (96/9/EC)	Protects substantial investment in the creation of databases.	Unauthorized extraction or reuse of a database may constitute a breach.
Computer Misuse Act	Penalizes unauthorized access to computer systems.	Bypassing security measures for scraping can lead to legal penalties.
General Data Protection Regulation (EU) 2016/679	Restricts the collection and processing of personal data without consent.	Personal data must not be scraped without adhering to GDPR requirements.

Table 2: Legal Considerations in Web Scraping

Table 2 shows the laws and their implication on web scraping. When developing a web scraper, one must make sure that these are taken into consideration.

2.5.2 Ethical Considerations

Respect for Robots.txt: A website's robots.txt file acts as a guideline for web crawlers and scraping programs. Scrappers should follow these guidelines ethically, even if they are not legally binding (Krotov, Johnson & Silva, 2020).

Bandwidth and Server Load: Scraping can put a lot of load on a website's server. Ethically, excessive scraping should be avoided because it may result in a denial-of-service problem for other users.

Transparency: Transparency on the data being gathered and its intended use is a must for ethical scraping. Businesses should be transparent with users on what practices they employ to obtain data.

Ethical Consideration	Description	Best Practices
Respect for Robots.txt	The robots.txt file provides guidelines intended to regulate the behavior of web crawlers.	Web scrapers should respect the instructions indicated in the robots.txt file of a website.
Bandwidth and Server Load	Scraping activities can consume significant bandwidth and processing resources of a web server.	Avoid high-frequency scraping that could disrupt the normal operation of the website.
Transparency	Being clear about the intent and use of collected data.	Disclose the intent of data collection and use to the data owners or through a public privacy policy.

Table 3: Ethical Considerations in Web Scraping

Table 3 presents a concise overview of key ethical considerations in web scraping, along with their descriptions and recommended best practices.

2.5.3 Guidelines for Ethical Scraping in Finland

In light of the complex legal and moral issues, the following recommendations are suggested regarding web scraping in Finland:

- ***Request Consent:*** Whenever feasible, obtain explicit permission from the website owner before scraping.
- ***Limit Data Extraction:*** Only gather information that is required and do not scrape personal information unless GDPR compliant.
- ***Respect Rate restrictions:*** To reduce server load, abide by any stated rate restrictions for webpage or API access.
- ***Clarify Your Intent:*** Be clear about who is gathering the data and why.

In Finland, web scraping must be approached with a great understanding of both legal and ethical constraints. While the legal system may appear to be complex, its primary goal is to safeguard copyright holders, database developers, and personal data. Ethical concerns, on the other hand, urge for respectful and transparent data gathering techniques that recognize online publisher's efforts as well as subject's data rights. A general overview is shown in Table 4.

Compliance Item	**Action Required**	**Notes**
Copyright adherence	Verify that the content is not copyrighted or that permission has been granted for its use.	Seek legal advice if unsure.
Database directive compliance	Ensure that database scraping doesn't infringe on the rights of the database maker.	Avoid large-scale extraction which can be seen as a 'substantial part' in the context of the directive.
Adhering to the Computer Misuse Act	Do not attempt to bypass any form of access control on the website.	Includes captcha solving, cookie manipulation, etc.
GDPR compliance	If personal data is collected, ensure there is a lawful basis for processing.	Consent, contractual necessity, legal obligation, vital interests, public task, legitimate interests.

Table 4: Compliance Considerations in Web Scraping

2.6 Challenges and Solutions in Web Scraping

2.6.1 Technical Challenges

Complex Anti-Scraping Techniques: Sophisticated anti-scraping features including dynamic content loading, IP filtering, and CAPTCHA have been used by websites, and they can greatly reduce the effectiveness of online scrapers (Turk, Pastrana & Collier, 2020). Although solutions like IP rotation and CAPTCHA cracking services have evolved, it remains a cat-and-mouse game between scraper developers and website administrators. To overcome this, the company may invest in advanced web scraping technologies that feature IP rotation and CAPTCHA solving capabilities, or it could seek partnerships with third-party services that specialize in overcoming such barriers.

Dynamic Content Loading Issues: Typical online scraping solutions cannot wait for content to load before scraping, which creates a unique difficulty when dealing with dynamic content loaded asynchronously via JavaScript (Turk, Pastrana & Collier, 2020). To get over this obstacle, we will look into utilizing headless browsers or other tools that can run JavaScript while they wait for the content to load.

2.6.2 Legal and Compliance Challenges

Navigating Data Protection Laws: The General Data Protection Regulation (GDPR) has set a high bar for personal data protection in Finland and the wider EU, complicating data gathering methods. It is imperative for researchers to comply with legislation around scraping, and anonymizing personal data is often required (EuropeanParliament, 2022). GDPR compliance is a must for Nettileasing Finland Oy. Implementing strong data anonymization and permission procedures, particularly when scraping user-related data, can assist the company in remaining in compliance with these rules.

Copyright and Intellectual Property Issues: The legal ramifications of online scraping are especially intricate in relation to information that is protected by copyright (Edwards & Veale, 2017). The literature generally advises using caution, respecting robots.txt files, and getting consent before scraping restricted information (Krotov, Johnson & Silva, 2020). it is important that when developing the web scraper for Nettileasing Finland oy, we strictly follow these rules.

2.6.3 Ethical and Privacy Challenges

Consideration for Personal Data and Privacy: Web scraping ethics extend beyond legal issues. There is a growing discussion about the moral responsibility researchers have when collecting potentially identifiable data. Anonymizing data and disclosing the usage of data

in a transparent manner are regarded excellent practices for mitigating privacy issues (Krotov, Johnson & Silva, 2020). Ethical web scraping procedures, such as anonymizing data and maintaining openness about data usage, are critical for Nettileasing Oy to retain public confidence and ethical standards.

2.6.4 Operational and Reliability Challenges

Maintenance and Adaptability of Scrapers: Due to the ever-changing nature of web page architecture, web scrapers are notoriously unstable. Ongoing maintenance is essential, and some literature advocates using machine learning to automatically adapt to structural changes (Kosala & Blockeel, 2000). Leveraging machine learning for the web scraper for Nettileasing oy will be out of the scope, we will be doing regular updates to the scraper. In future when the web scraper scales, then we might want to look into integrating machine learning techniques to adapt to the ever changing nature of websites.

Ensuring Data Integrity: It is crucial to maintain the integrity of the data that is gathered via scraping. To guarantee the integrity and dependability of data utilized for analysis, literature stresses the significance of putting strong data validation and cleaning procedures in place (Ridzuan & Wan Zainon, 2019). Implementing comprehensive data validation and cleaning methods is critical to ensuring the quality and reliability of scraped data, which has a direct influence on Nettileasing Finlands Oy's decision-making and customer satisfaction.

2.6.5 Scalability and Infrastructure Challenges

Demands of Scalability: As the volume of data and frequency of scraping operations rise, so does the necessity for scalable solutions in web scraping. To satisfy these expectations, the literature recommends distributed scraping systems and cloud-based solutions (Chaulagain et al., 2017). As Nettileasing Finland Oy expands its operations, distributed scraping systems and cloud-based solutions will be required to properly handle the increased data volume.

Managing Large Volumes of Data: The large volume of data included in scraped datasets demands advanced storage and processing methods. To successfully manage enormous amounts of data, studies have advised the adoption of big data frameworks and sophisticated database systems (Taleb et al., 2021). In our initial scope of the project, the data we will be collecting will not be as large to require these measures, but as it scales then we will look into integrating big data frameworks.

Challenge Category	Specific Challenge	Solution
Technical	Anti-Scraping Techniques	IP rotation, behavioral mimicry, randomized requests
Technical	AJAX-Loaded Data	Headless browsers, direct AJAX parsing
Legal	Data Protection Regulations	Legal review, GDPR compliance
Legal	Copyright Infringement	Focus on public data, obtain permissions
Ethical	Personal Data Anonymity	Data anonymization techniques, secure handling practices
Operational	Scraper Maintenance	Machine learning for adaptation, regular updates
Operational	Data Fidelity	Data validation processes, anomaly detection
Scalability	High-Performance Infrastructure	Distributed computing

Table 5: Challenges and Solutions in Web Scraping

From Table 5, It is clear that, even if web scraping is a useful tool for gathering data, its proper implementation necessitates careful consideration of operational, ethical, legal, and technological considerations.

2.7 Scaling Web Scraping and Community Support

2.7.1 Scalability in Web Scraping

Understanding Scalability Challenges: Scalability is not just a technological challenge but also a strategic one in the web scraping domain. Large dataset management, evading detection by anti-scraping methods, and sustaining performance consistency across many scales are some of the issues that our data scraping processes might face as they grow in scope.

Strategies for Scalability: Effective web scraping scalability solutions tend to rely upon distributed computing, cloud-based resources, and asynchronous processing. These techniques provide simultaneous data processing and robustness against potential website blockages.

2.7.2 Distributed Web Scraping

The use of many workstations to distribute the workload effectively mitigates the risks associated with single-point scraping activities, such as IP blacklisting and rate limitations. To put such a system in place for our project, a careful balance of task allocation, error management, and data synchronization is required. The main obstacles in distributed web scraping include maintaining a bigger infrastructure, inter-node communication, and effective resource use.

2.7.3 Cloud Services and Scalability

Cloud platforms such as AWS, Azure, and Google Cloud allow web scrapers to expand on-demand, pay for just the resources utilized, and have access to a diverse set of tools built to handle large-scale data processing jobs. Although this was our initial plan for our project, due to some technical and financial challenges that will be discussed later, hindered our ability to implement a cloud based solution. But a plan is still in place to implement a cloud based scraping solution in the future.

2.7.4 Open Source Tools and Community Engagement

Scrapy and Scrapy Cluster: The well-known open-source framework Scrapy is made for effective and quick data extraction. Scrapy Cluster, an extension, enables the distribution of scraping tasks over multiple machines, increasing the framework's scalability for large-scale applications (Kouzis-Loukas, 2016).

Beautiful Soup and Concurrent Requests: Beautiful Soup may be used in conjunction with concurrency libraries like as grequests or aiohttp to increase the number of simultaneous requests even though it is not designed for large-scale distributed scraping (Richardson, 2023).

Selenium Grid: Selenium Grid provides a scalable environment for sophisticated scraping operations that involve rendering JavaScript by enabling browser-based scraping simultaneously across several workstations and browsers (Selenium, 2023).

Playwright: Playwright is a relatively new tool in the web scraping space. It can be deployed across cloud services and is very scalable since it supports headless browser instances and can be containerized within Docker (Lekh, 2023).

Tool	**Parallel Execution**	**Cloud Integration**	**Distributed Architecture**	**Community Support**
Scrapy	Native	Yes	Yes (with Scrapy Cluster)	Extensive
Beautiful Soup	Via external libs	Yes	No	Moderate
Selenium Grid	Yes	Yes	Yes	Extensive
Playwright	Yes	Yes	Yes	Growing

Table 6: Comparison of Open-Source Web Scraping Tools

The literature emphasizes how crucial scalable web scraping technologies are in the data-driven world of today. The scalability of scraping technologies is crucial for organizations and researchers that aim to extract insights from large volumes of online data.

Open-source solutions such as in Table 6 are leading the way in innovation when it comes

to community support because of their collaborative nature. The community offers a wealth of plugins, user-contributed code snippets, and comprehensive documentation, all of which may be quite helpful for tailoring scraping solutions to meet scaling needs. The literature also shows a strong trend toward distributed, cloud-based, and community-supported web scraping solutions at scale. Open-source tools and frameworks continue to evolve with community feedback playing an important role. As the volume of web data grows at an exponential rate, the scalability of web scraping processes remains an important field of research and development.

In conclusion, the literature review delves deeply into the evolution and present state of web scraping in the context of e-commerce, with a specific emphasis on its application for Nettileasing Finland Oy. The review opens with a history of e-commerce, highlighting the growing importance of data in determining customer experiences and company strategies. This context is essential for understanding the importance and necessity of web scraping tools and methods in today's data-driven e-commerce industry.

The review explores the technical details of web applications, providing a solid understanding of client-side and server-side components, which is critical for the design and implementation of efficient web scraping strategies. This technological evaluation is particularly relevant to the desktop application developed for this thesis, which makes use of Scrapy and Playwright. It emphasizes the significance of selecting proper tools for various scraping situations particularly given the problems caused by dynamic content and the requirement for JavaScript execution capabilities.

The thorough review of web scraping technologies and tools covers a wide range of solutions, including browser automation tools like Selenium and Playwright as well as Python-based tools like BeautifulSoup and Scrapy. This section not only describes the features of these tools, but it also includes real-world applications and case studies that show how they may improve data collection accuracy and efficiency in e-commerce. These insights directly support the research question regarding the implementation of an automated web scraping system and its impact on the accuracy and efficiency of e-commerce data collection for Nettileasing Finland Oy.

The review also explores the legal, ethical, and operational challenges surrounding web scraping. It gives an in-depth understanding of the complexity involved, particularly in the context of GDPR compliance and ethical issues such as robots.txt file compliance and server load. This section of the study emphasizes the need of establishing web scraping solutions that are not only technically sound but also legally and ethically responsible, which is critical for Nettileasing Finland Oy's project's success.

Furthermore, the literature review discusses web scraping scalability and community support. It examines the strategic relevance of scalability in web scraping, particularly for a young a company like Nettileasing Finland Oy, as well as how distributed computing, cloud-based resources, and community-driven open-source tools may manage these challenges effectively. The review highlights the critical role of community support and open-source tools in driving innovation and providing scalable solutions, aligning with the project's use of Scrapy and Playwright and its future scalability plans.

Overall, this literature review provides a good framework for the next sections of this thesis. It offers a full and comprehensive examination of web scraping and data collection in the context of e-commerce, highlighting both the technical and strategic aspects of this domain. The findings of this evaluation will be used to influence the approach and execution of Nettileasing Finland Oy's web scraping system, ensuring that the project not only meets but surpasses its research objectives of enhancing data collecting accuracy, efficiency, customer satisfaction, and retention.

3 Methodology

3.1 Research Design

This thesis employs the 'Research Through Design' (RtD) technique, which is ideal for projects that combine technological innovation with theoretical research. The focus of RtD is on implementing and researching technical artifacts, with the design process itself serving as a tool of inquiry and knowledge acquisition. The artifact in the context of this project is a desktop web scraping software that was carefully designed to satisfy the unique data gathering needs of Nettileasing Finland Oy. This approach goes beyond simply creating a useful tool, it investigates the ways in which this technology could transform e-commerce industry processes, specifically with regard to data collection, accuracy, and efficiency. The RtD method allows for an in-depth dive into both the practicalities and theoretical underpinnings of web scraping technology, offering a detailed overview of its applications and implications.

Iterative Design Process

The web scraping system was developed using an iterative approach, which is a key component of the RtD methodology. This process involved multiple stages:

- ***Identifying Requirements:*** Initially, the project involved defining Nettileasing Finland Oy's specific data gathering requirements. This stage was all about determining the functional and technical requirements for the application.
- ***Prototype Development:*** Following the collection of requirements, early prototypes of the web scraping application were developed. These prototypes were used to test the viability of the design concepts and technology used.
- ***Testing and Feedback:*** Each prototype was rigorously tested in simulated e-commerce environments to ensure its functionality. The results of these tests were crucial in determining areas for improvements, such as data extraction accuracy, processing speed, and overall system stability.
- ***Refinement and Iteration:*** Testing results were utilized to improve the application. This iterative process of creation, testing, and refining was repeated until the application satisfied the necessary performance and usability criteria.

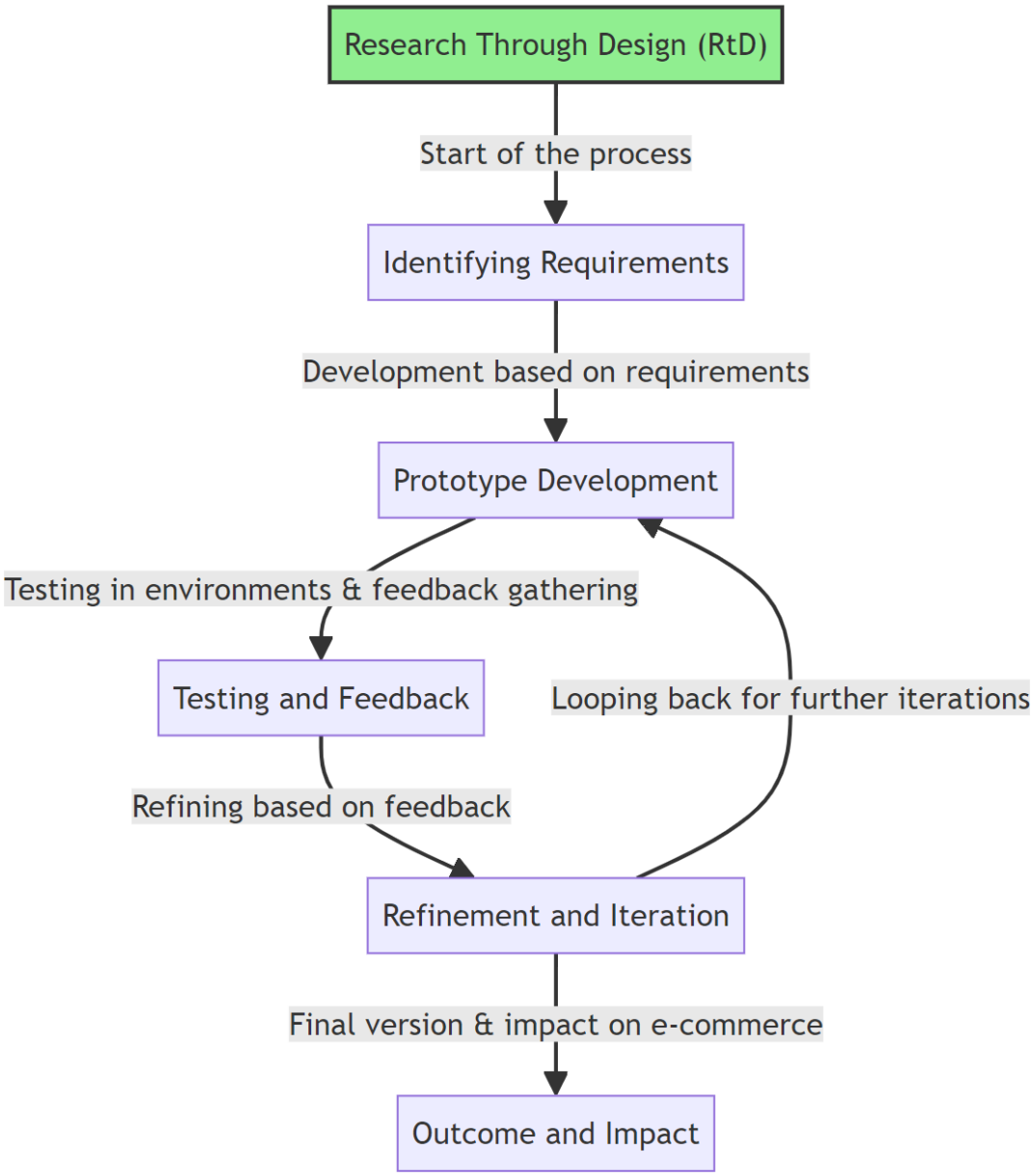

Figure 1: Research Through Design (RtD) Process

Each stage, as shown in 1, of this iterative process was properly documented, and the results were rigorously examined to verify that the progress was in line with the research objectives. Addressing the research questions required iterative design and development of the web scraping system. The practical insights gathered during the application's development phases were critical in investigating the efficacy and challenges of automated web scraping in the context of e-commerce, thereby answering the first research question. Furthermore, the application's continuous improvements and refinements gave preliminary data on its potential to boost customer happiness and retention, which is relevant to the second study topic.

3.2 Data Collection Techniques

Data collection in this project was primarily executed through the custom-developed web scraping application. The application was designed to specifically target and extract data from a selected set of e-commerce websites with which Nettileasing Finland Oy had established agreements for data scraping. These website were:

- tietokonekauppa.fi
- venekauppa.com/fi
- bikemarine.fi

The scope of data collection was comprehensive, encompassing various product attributes such as:

- Category name
- Sub-category name
- Product title
- Product link
- Product images
- Brand name
- Model number
- Product descriptions
- Monthly price detail
- Leasing price detail
- Number of payments

This tailored strategy to data collection was in direct response to the project's objective of improving efficiency and accuracy in acquiring e-commerce data.

3.3 Data Analysis Methodology

The data collected via the web scraping application was subjected to a thorough analytical process to evaluate its accuracy and efficiency. This analysis involved several steps:

- ***Data Completeness:*** Assessing whether all necessary data points were successfully extracted from the target websites.
- ***Accuracy Assessment:*** Comparing the extracted data against the actual information

on the e-commerce sites to gauge the precision of the scraping process.

- ***Efficiency Evaluation:*** Measuring the time taken for data collection and processing, and evaluating this against the efficiency benchmarks established in the project requirements.

For these analyses, a range of statistical tools and comparison metrics were employed. This quantitative approach ensured that the performance evaluation of the application was objective, reliable, and aligned with the research questions.

3.4 Tools and Technologies Used

A lot of consideration went into choosing the right tools and technologies while developing the web scraping system for Nettileasing Finland Oy. The success of the project depended on selecting a set of technologies that were not only reliable and flexible but also specifically designed to address the specific challenges associated with web scraping in a dynamic e-commerce environment. Having carefully considered all of the possibilities, and making sure they met the project's needs, the Scrapy framework and Playwright were chosen as the main tools for this project.

3.4.1 Scrapy Framework

The open-source, well-established Python web scraping framework Scrapy was selected due to its robust scraping features and widespread adoption in the data extraction field. Important elements that influenced its choice.

Speed and Efficiency: Scrapy is renowned for its lightning-fast speed, which is essential for managing the massive amounts of data that are commonly seen on e-commerce websites.

Asynchronous Processing: It can process requests asynchronously, which makes it extremely effective in terms of network utilization and speed, which is critical for real-time data scraping.

Extensibility: Scrapy offers the flexibility required to adjust the scraping process to the complex requirements of various e-commerce platforms thanks to its built-in support for expanding its capabilities through custom middlewares and pipelines.

Robust Error Handling: Its advanced error tracking and handling systems guarantee the reliability of the scraping process and enable prompt identification and resolution of problems.

These features make Scrapy an ideal choice for our project.

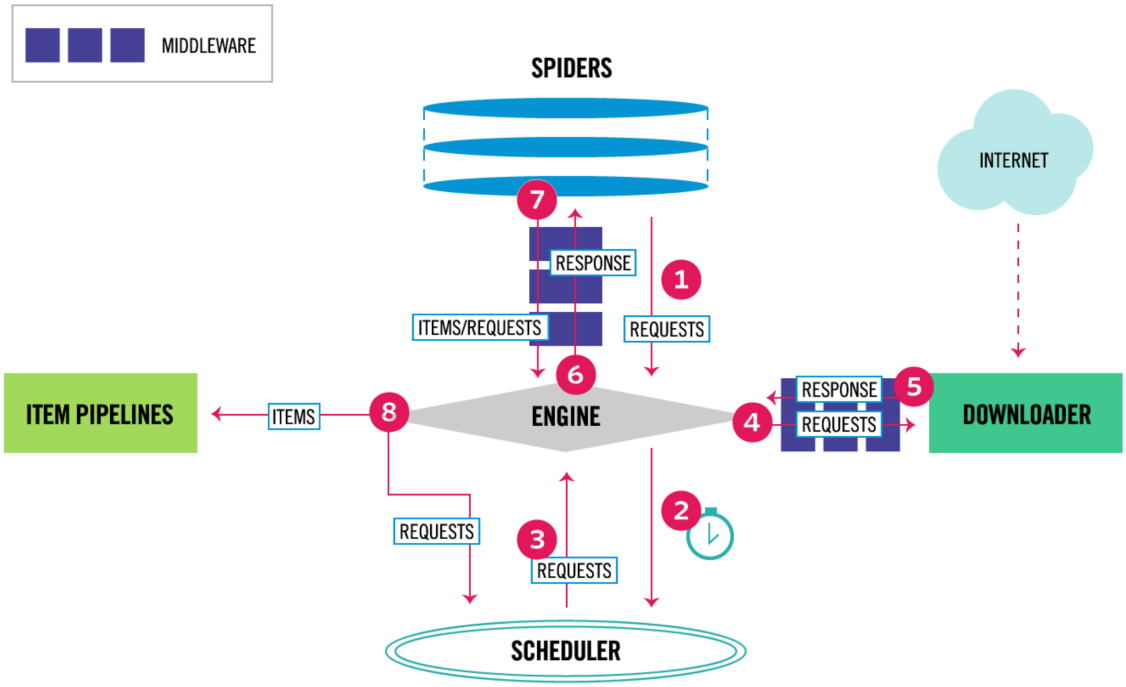

Figure 2: Scrapy Architecture (Scrapy, 2023)

Figure 2, provides an in-depth look at the Scrapy framework's architecture, highlighting its modular design where each component is tasked with specific function. This modular approach enhances testability and maintainability of the scraper.

Engine: The engine acts as the core of the Scrapy framework, orchestrating the workflow among various components. It has a critical role in managing event triggers and responses, including handling scenarios like request errors, response mishaps, and exceptions. This central unit ensures smooth communication and process flow within the Scrapy framework. (Scrapy, 2023)

Scheduler: The Scheduler plays a pivotal role in task management, deciding the execution timing of tasks based on an organized queue system. It has the capability to control the delay between requests, thereby optimizing the scraping process efficiency and resource allocation. (Scrapy, 2023)

Downloader: The Downloader component is important for executing HTTP requests. In standard operations where a real browser is not utilized, it retrieves and forwards HTTP response contents back to the engine. However, in scenarios employing a real browser for requests, this component is substituted by a specialized middleware capable of browser control, thereby adapting to different operational contexts. (Scrapy, 2023)

Spiders: Spiders, crafted by developers, are crucial for dictating the scraping actions to acquire and parse specific web content. They offer a platform for setting custom configurations for the Downloader and its associated middlewares. The output from these spiders, typically parsed web content, is subsequently relayed to the Item Pipeline for further processing. (Scrapy, 2023)

Item Pipeline: This component processes the data returned by Spiders. It is responsible for tasks such as data validation, custom transformation, cleansing, and ultimately storing the processed data in various Data Storage systems (e.g., Redis, MongoDB, Postgres). This pipeline ensures data integrity and usability in subsequent stages or applications. (Scrapy, 2023)

Downloader Middlewares: Functioning as an intermediary, these middlewares manage both inbound and outbound traffic between the Downloader and other components. They enhance requests and responses with custom metadata, and offer a venue for developers to implement specific rules, like retry mechanisms, thus enhancing the robustness of the scraping process. (Scrapy, 2023)

Spider Middlewares: Similar to their Downloader counterparts, Spider middlewares operate between the Spider and Engine. They provide a layer where developers can perform custom preprocessing on Spider inputs, such as modifying target URL parameters, thereby adding a layer of flexibility and control over the scraping process. (Scrapy, 2023)

Each of these components works in tandem, ensuring that Scrapy remains a powerful and flexible tool for web scraping tasks, characterized by its ease of testing and maintenance.

3.4.2 Playwright

Playwright is a newer automation framework, but it was chosen for its ability to handle dynamic JavaScript-heavy content, which is becoming more common in current web applications. Reasons for its selection include:

Browser Automation: Playwright provides a high-level API for automating browser sessions, allowing simulation of user interactions on web pages, required to trigger dynamic content loading.

Support for Headless Browsers: It supports headless browsers, which means it can carry out web scraping tasks without the overhead of a user interface, which is useful for server-side automation and batch processing.

Cross-Browser Compatibility: Playwright enables scripts to run in multiple types of browsers, ensuring compatibility and reliable scraping results across different web environments.

Modern Web Support: It is built to handle the most recent web technologies, making it an ideal tool for scraping data from websites that use a lot of client-side scripting.

Scrapy and Playwright work together to deliver a full web scraping solution. Scrapy excels in extracting data from static pages and navigating complicated website architectures, whereas Playwright is useful in situations where client-side scripting would otherwise obscure the data extraction process. These solutions work well together to satisfy the project's requirements for speed, accuracy, and adaptability in data collection from various e-commerce sources.

4 System Design and Implementation

4.1 System Architecture and Design

4.1.1 General Overview

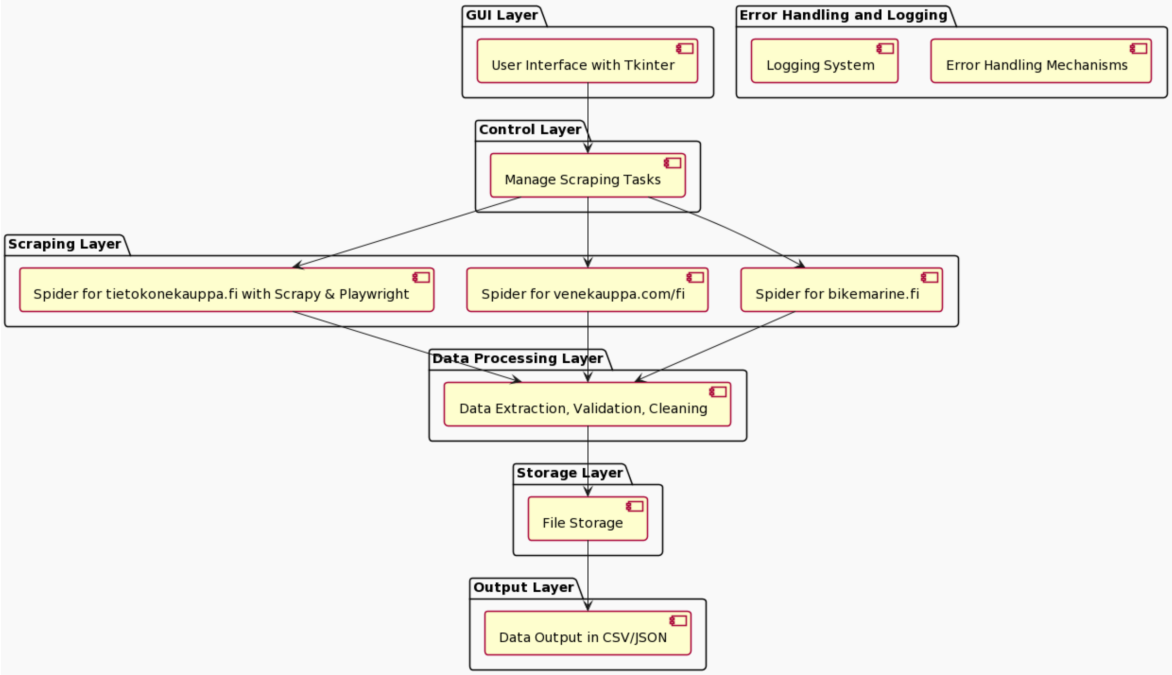

Figure 3: Nettileasing's Web Scraping System Architecture Diagram

Figure 3, shows the architecture for Nettileasing web scraping application, which is based on a modular design that is developed to effectively manage the extraction of data from three different websites. At the foundation of this architecture is the Scrapy framework, which is handling all of the fundamental web scraping tasks. The system is designed to be modular and scalable, enabling future expansions or changes to be made with minimum disruption to the current system.

Several spiders communicate with the Scrapy core, each built to target a specific e-commerce website. These spiders are the system's backbones, crawling through web sites, sending requests, and processing HTML content to extract data. The extracted data is then processed and validated through a number of pipelines to guarantee quality and consistency before being saved or used.

A standalone graphical user interface (GUI) supplements the system by allowing end users to control and monitor scraping activity. The GUI allows users to start/stop scraping jobs, change settings, and see logs that show current status and progress.

4.1.2 Scrapy Framework

Scrapy is an open-source framework that is commonly used to provide the infrastructure required for web crawling and scraping. It is selected due to its extensive feature set, which includes integrated support for data extraction, processing, and request management.

Each spider in the Scrapy project is designed to scrape a single e-commerce site. These spiders identify the data to be extracted using the definitions in the items.py module. They operate together with Scrapy's middleware and item pipelines to improve the scraping process:

Middleware: These components improve scraping operations by managing cookies, redirecting failed requests, and evading detection by web servers.

Item Pipelines: Once the data has been retrieved, it is routed through a number of pipelines that clean, validate, and store it, ensuring that it matches the specified standards and is ready for use.

4.1.3 Playwright Integration for Dynamic Content

To scrape tietokonekauppa.fi, which needs interaction with JavaScript for pagination, Playwright is incorporated into the Scrapy framework to run JavaScript and fully render the pages and do pagination before data extraction.

The spider is set up to launch a Playwright browser instance, navigate to the desired URLs, and wait for the JavaScript to execute. The spider then extracts data after rendering the content.

```
# Code Snippet
# Start the Playwright tool.
self.playwright = sync_playwright().start()
self.browser = None  # Initialize browser as None

def launch_browser(self):
    # Only launch if browser isn't already open.
    if not self.is_browser_open():
        custom_print("Browser Launched")
        # Start a new browser instance.
        self.browser = self.playwright.chromium.launch(headless=True)
```

A custom Scrapy middleware controls the invocation of the Playwright browser, ensuring that it is used only when necessary. When scraping tietokonekauppa.fi, this middleware intercepts Scrapy's request processing and switches to a headless browser.

```
# Code Snippet
class PlaywrightMiddleware:
    @classmethod
    def from_crawler(cls, crawler):
        middleware = cls()
        crawler.signals.connect(middleware.spider_opened, signal=signals.
            spider_opened)
        return middleware

    def spider_opened(self, spider):
        # Initialize the playwright page attribute in spider
        spider.playwright_page = None

    def process_request(self, request, spider):
        if spider.name == 'tietospider':
            return PageCoroutine('goto', request.url, wait_until='
                networkidle')
```

To facilitate the integration and ensure a seamless operation of playwright, several libraries and custom-written code are employed:

Playwright's Asynchronous Nature: Playwright uses an asynchronous operating framework that is controlled by the Scrapy event loop. This assures that Playwright's non-blocking behavior is consistent with asynchronous architecture of Scrapy.

Error Handling: Robust error handling mechanisms are in place to address any issues arising from the integration, such as navigation timeouts or rendering errors.

Performance Considerations: Performance optimization has been applied, minimizing the overhead caused by browser automation. This involves configuring the browser to make scraping jobs more efficient and using Playwright only on pages where it is really essential.

Scrapy integration with Playwright reflects a hybrid approach to web scraping, combining the strengths of both frameworks to manage the wide range of content delivery techniques used by modern e-commerce sites. This combination assures that the system is not only adaptable to different types of websites, but also efficient and reliable in its operation.

4.2 Project Structure

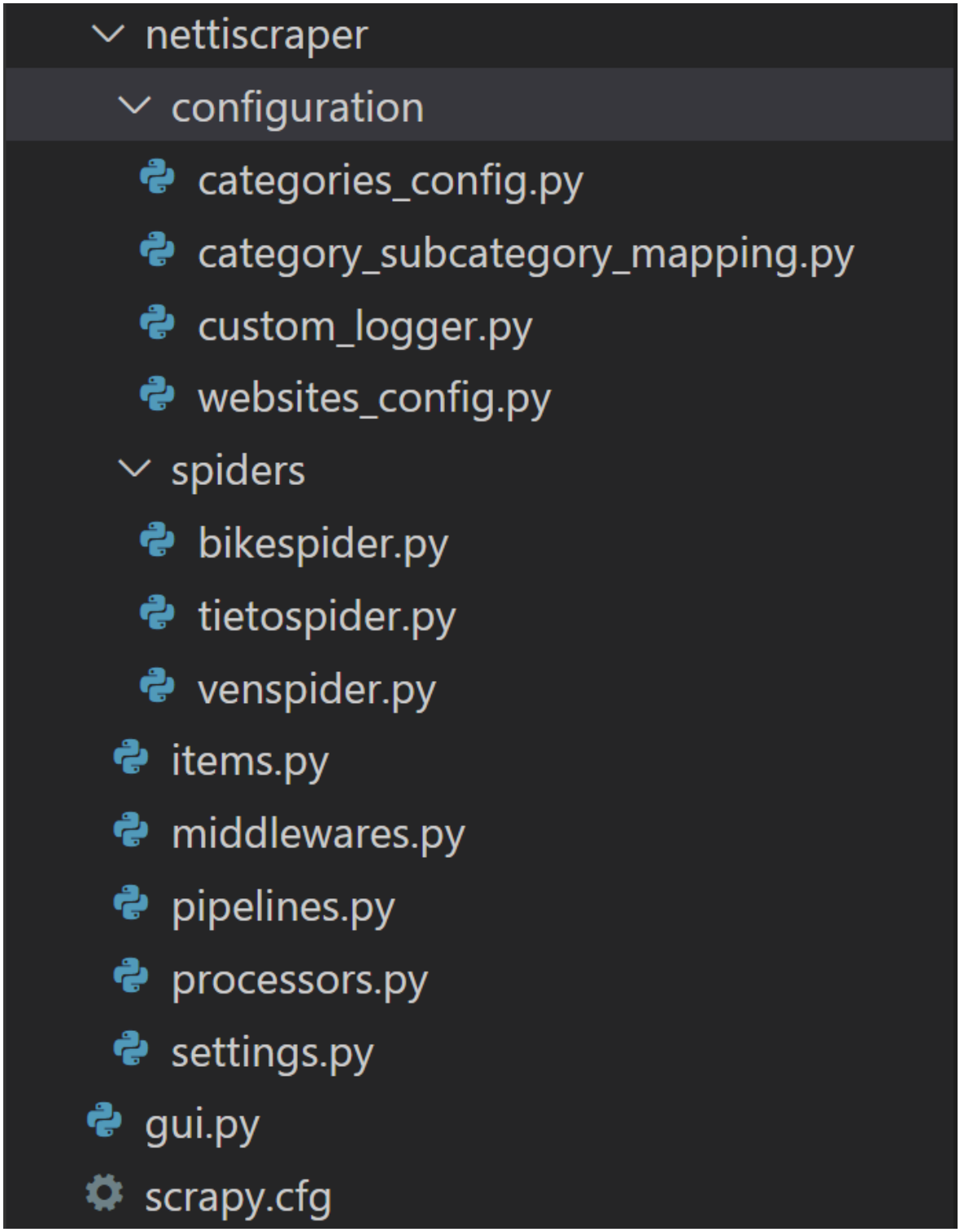

Figure 4: Project File Structure in VS Code

Figure 4, show how the project is structured in VS code. Here are the details on each directory and file.

4.2.1 Spiders Directory

bikespider.py: The spider script dedicated to scraping information from the bikemarine.fi website.

tietospider.py: The spider script dedicated to scraping information from the tietokonekauppa.fi website.

venspider.py: The spider script dedicated to scraping information from the venekauppa.com website.

The spiders in the Spiders directory uses shared components inside the Configuration Directory and Root Directory. These shared procedures built across all spiders greatly improve the efficiency and maintainability of the web scraping system. These components guarantee that common features are abstracted, minimizing redundancy and making the addition of new spiders and the maintenance of current ones easier.

4.2.2 Configuration Directory

categories_config.py: This file contains configurations for different websites that the web scraper will target. Each website has a list of categories, and each category has a list of subcategories. The spiders will use these configurations to navigate through the websites and extract relevant data.

category_subcategory_mapping.py: It contains 'mappings' which is a dictionary that holds the renaming configurations for categories and subcategories for different websites. As the data is extracted from each website, the category and subcategory names are renamed according to the names on nettileasing website. This is done to ensure that when we upload the data into the nettileasing's database, each product is uploaded under the correct category and subcategory.

custom_logger.py: Custom logging configurations are defined here. This file handles the logging of events, errors, and other runtime information from the spiders, customizing how logs are generated and where they are stored.

websites_config.py: This file holds the configurations specific to different websites being scraped, such as base URLs, allowed domain and website specific CSS and XPATH selectors.

4.2.3 Root Directory of nettiscraper

items.py: Defines the data models for the items that the scrapers will collect. It structures the scraped data by setting fields for items like product names, prices, images, and more. The items.py module plays a pivotal role in the system. It defines the data structure for items that the spiders will extract. This shared item definition ensures that all spiders extract and structure data in a uniform way, which simplifies data cleaning, validation, and storage. The ProductDetailItem class in items.py acts as a template for the scraped data, ensuring the standard format that the system expects.

```
# Code Snippet
# Defining a class `ProductDetailItem` that inherits from `scrapy.Item`.
class ProductDetailItem(scrapy.Item):
    main_category_name = scrapy.Field()
    subcategory_name = scrapy.Field()
    product_url = scrapy.Field()
    product_images = scrapy.Field()
    title = scrapy.Field()
    brand = scrapy.Field()
    model = scrapy.Field()
    description = scrapy.Field()
    monthly_price = scrapy.Field()
    leasing_price = scrapy.Field()
    location = scrapy.Field()
    number_of_payments = scrapy.Field()
```

middlewares.py: Contains the middleware components for the Scrapy framework.

```
# Code Snippet
SPIDER_MIDDLEWARES = {
    "nettiscraper.middlewares.NettiscraperSpiderMiddleware": 543,
}
```

pipelines.py: Defines the item processing pipelines. After an item is scraped, it is passed through a pipeline where it can be cleaned, validated, and stored or exported. We have created a custom pipeline processors.py to do this task.

processors.py: Defines a custom item processing pipeline. The “processors.py” module’s data processing pipelines are essential for guaranteeing that data from various spiders is handled consistently. These pipelines handle operations including data cleaning, validation, and transformation. For example, the ProductDetailLoader class in processors.py offers a set of input and output processors that standardize the cleaning and formatting of data fields, such as converting strings to numerical values or combining multiple image URLs into a single string. By centralizing these tasks, we can ensure data integrity and uniformity across

all scraped content.

```
# Code Snippet
def clean_description(description):
    # Split the description into lines and strip whitespace from each
        line
    lines = [line.strip() for line in description.splitlines()]
    # Remove empty lines
    lines = [line for line in lines if line]
    # Join the lines back into a single string with a single newline
        between each line
    cleaned_description = "\n".join(lines)
    return cleaned_description

class ProductDetailLoader(ItemLoader):
    default_input_processor = MapCompose(clean_data, remove_tags)
    default_output_processor = TakeFirst()

    # For fields that return multiple values and need to be joined
    product_images_out = Join(",")

    # For fields that need to be converted to float
    monthly_price_in = MapCompose(clean_data, to_float)
    leasing_price_in = MapCompose(clean_data, to_float)

    # For fields that need to be converted to int
    product_code_in = MapCompose(clean_data, to_int)
    model_in = MapCompose(clean_data, to_int)
    number_of_payments_in = MapCompose(clean_data, to_int)
```

settings.py: Contains settings for the Scrapy project. This include configurations like concurrent requests limits, user-agent strings, and other operational parameters. The configuration parameters that control how each and every spider behaves are stored in the settings.py module. These customizations include download delays, concurrency limits, bot identification (such as the USER_AGENT string), and other operational properties. We make sure that all spiders follow the same operating standards by combining these parameters, which is important for both controlling the load on our scraping infrastructure and obeying the guidelines of target websites.

gui.py: The Python script for the graphical user interface (GUI) using Tkinter. It include code for creating windows, buttons, and other UI elements, allowing users to interact with the scraping system. While each spider has its own scraping algorithm, the GUI serves as a single point of interface for the user to control and monitor the scraping operation. The GUI script interacts with the spiders by invoking them and providing real-time feedback on

scraping progress. It provides a user-friendly interface for controlling the scraping system while abstracting the complex nature of the underlying scraping algorithm.

4.3 Website-Specific Implementation

Across the scraping system, all spiders follow a consistent design pattern established within the Scrapy framework. This includes initializing the scraping process, navigating to targeted subcategories, and parsing product listing pages. The spiders leverage Scrapy css and xpath selectors for extracting data, such as titles, images, descriptions, and pricing information. Error handling is a crucial part of the design, with mechanisms in place to manage network errors, missing content, and other issues that may arise during scraping. While the general operation of the spiders is consistent, each spider is tailored to the specific challenges and structures of the individual websites it targets. Also we have specific categories and subcategories for each website from which we need to scrap product data. These categories and subcategories are defined in categories_config.py.

To implement any web scraper, it is important to learn about the target website structure and robots.txt. Also we need to keep in mind that the contents in robots.txt and the structure of the websites can change over time.

Before we design our spider for target website, we selected a product category and subcategory from the website and opened the Inspect tool from the developer tools in google chrome and stared to check where our data point are in the HTML. We checked weather the target websites are displaying content in a static format or uses JavaScript rendering. We checked this by disabling JavaScript in our browser and reloading the page and checking if there are any changes to the content of the page.

4.3.1 Scraper Implementation for bikemarine.fi (bikespider)

Website Investigation and Planning

Before we start to design and implement our scraper for bikemarine.fi, we need to check its robots.txt file and check if there is any restriction placed by the website for scraping bots or if there is a restrictions on specific data points being scraped.

```
User-agent: *
Disallow: /wp-admin/
Allow: /wp-admin/admin-ajax.php
```

Figure 5: robots.txt from bikemarine.fi

Based on the Figure 5, it instructs all web crawlers (User-agent: *) not to access or index the admin area of the site (Disallow: /wp-admin/). However, it makes an exception, allowing these crawlers to access a specific file used for handling Ajax requests (Allow: /wp-admin/admin-ajax.php). Most importantly, it also does not disallow scraping product data from the website.

Website Structure

The first step is to get all the product links from each subcatgeory. We open a subcategroy from the website and then run our inspect tool and check where these product links are located.

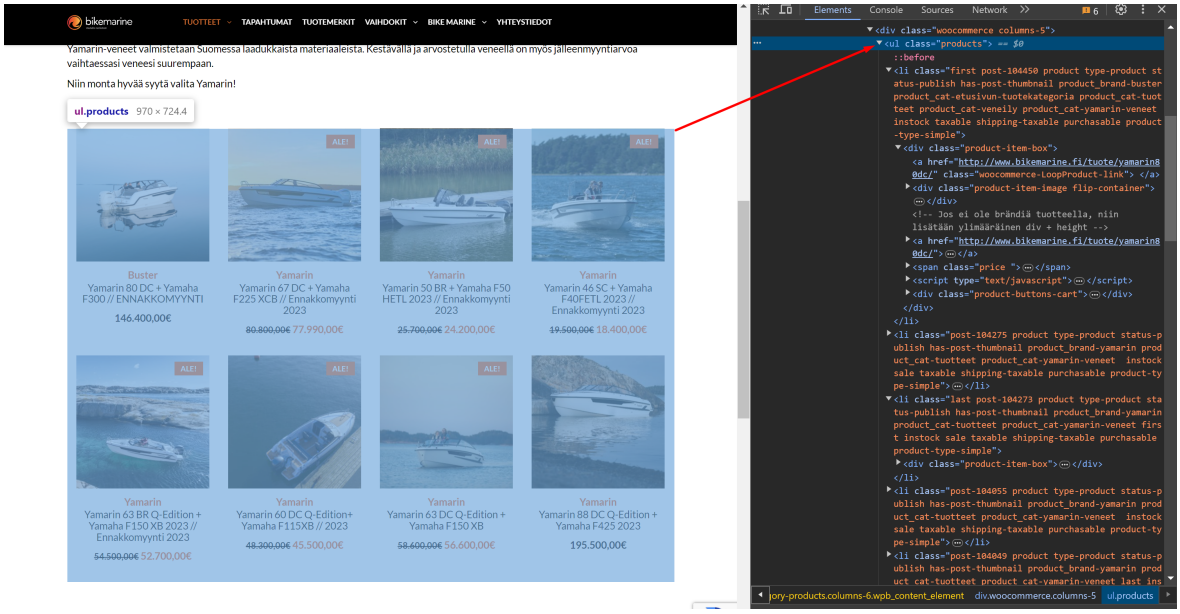

Figure 6: Location of Product Links on bikemarine.fi

We can see in Figure 6, that all the product in a subcategory are in a <ul class="products"> tag. Each product is represented by <li> tag within <ul> tag. Each tag within <li> tag represents information about specific product, including its links which is in <a href""> tag. We will use this information to build our selectors to get product links.

Now the second step is to get product details for each product. For this we will investigate each product page.

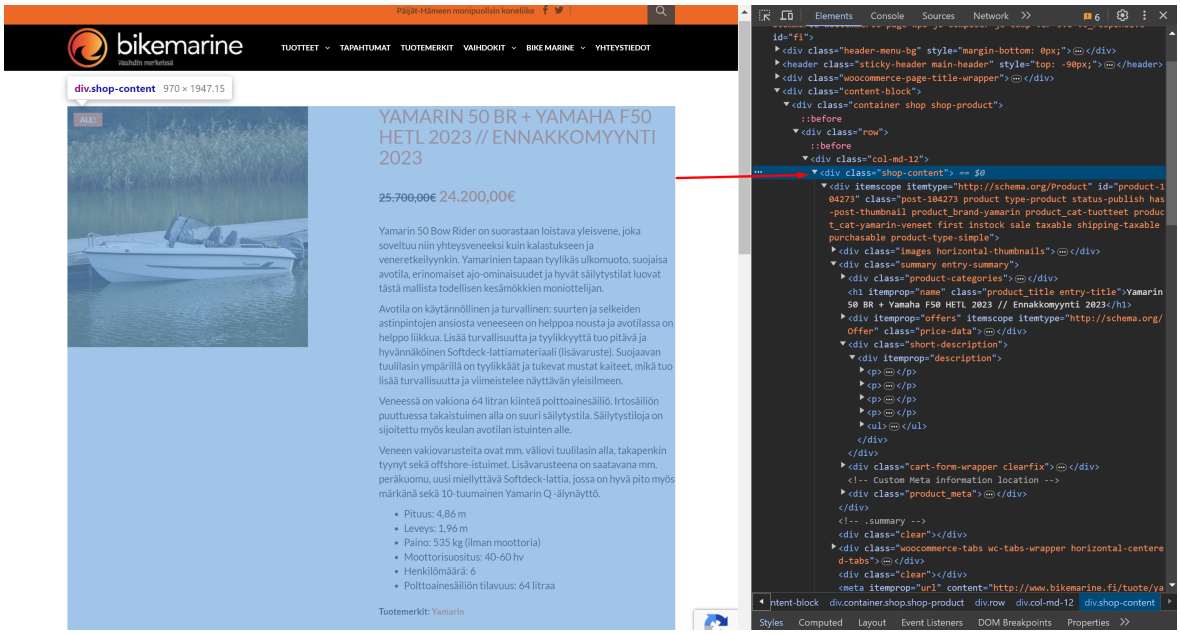

Figure 7: Location of Product Details on bikemarine.fi

From Figure 7, we can see that all information about the product, title, images, description, price, etc. are inside <div class“shop-content”> tag. We further investigated this to get specific location of each data point required for our project.

The bikemarine.fi website presents product information in a static format, which the bike-spider navigates without the need for JavaScript rendering. Its scraping process is straightforward, using direct Scrapy CSS and xpath selectors.

Scraper Implementation

After instigating the website and locating our data points, we defined our css selectors in website_config file. These selectors as shown in Figure 8, will be used by the spider for bikemarine.fi.

```
# A dictionary of CSS selectors that are used to extract specific data from the website.
"selectors": {
    "product_links": "ul.products div.product-item-box > a:nth-child(1)::attr(href)",
    "title": ".product_title.entry-title::text",
    # "product_code": "div.ourcode::text",
    "product_images": [
        "#product-main-image::attr(href)",
        ".wpb_wrapper .wpb_single_image .vc_single_image-wrapper > img::attr(src)",
    ],
    "brand": ".product_meta .wb-posted_in a::text",
    "description": ".short-description div[itemprop='description'] p::text",
    "monthly_price": [
        "p.price ins .woocommerce-Price-amount::text",
        "p.price .woocommerce-Price-amount::text",
    ],
    # ... other CSS selectors specific to bikemarine
},
```

Figure 8: CSS Selectors for bikemarine.fi

Data Collection Process

The BikespiderSpider initiates the data collection process with the start_requests method, which marks the beginning of the scraping process. This method sets the starting point by selecting the bikemarine.fi website and a predefined list of categories and then calls the bikemarine_requests function to start scraping.

```
# Code Snippet
def start_requests(self):
    self.start_time = time.time()
    custom_print(f"Spider started at {time.ctime(self.start_time)}")
    custom_print(f"Current Website: bikemarine.fi")
    self.current_website_index = 0
    website = CATEGORIES[self.current_website_index]
    yield from self.bikemarine_requests(website)
```

The bikemarine_requests method sends HTTP requests to each subcategory within the bikemarine.fi website. It uses Scrapy's Request function to asynchronously fetch the page content.

```
# Code Snippet
def bikemarine_requests(self, website):
    for category in website["categories"]:
        for subcategory in category["subcategories"]:
            yield scrapy.Request(
                subcategory["subcategory_link"],
                callback=self.bikemarine_parse_products_links,
                meta={"subcategory": subcategory, "category_name":
                    category["category_name"], "website": website}
            )
```

Upon receiving a response, the bikemarine_parse_products_links method parses the HTML content to extract product links using CSS selectors. These selectors are configured specifically for the bikemarine.fi website.

```
# Code Snippet
def bikemarine_parse_products_links(self, response):
    selectors = WEBSITE_CONFIGS[response.meta["website"]["website"]]["
        selectors"]
    products = response.css(selectors["product_links"]).getall()
    for product_link in products:
        yield response.follow(
            product_link,
            self.bikemarine_parse_product_details,
            meta={"subcategory": response.meta["subcategory"], "
                category_name": response.meta["category_name"], "website":
                 response.meta["website"]}
        )
```

Data Processing and Loading

For each product link, the bikemarine_parse_product_details method is invoked. This method is the core of the data extraction process, where detailed information about each product is obtained.

```
# Code Snippet
def bikemarine_parse_product_details(self, response):
    loader = ProductDetailLoader(item=ProductDetailItem(), response=
        response)
    loader.add_value("main_category_name", response.meta["category_name"
        ])
    loader.add_value("subcategory_name", response.meta["subcategory"]["
        subcategory_name"])
    loader.add_value("product_url", response.url)

    # Extracting various product details like title, images, brand, and
        description
    # Example of Extracting the product title.
        try:
            title_extracted = response.css(selectors["title"]).get()
            if title_extracted:
                loader.add_value("title", title_extracted.strip())
            else:
                custom_log_error(f"No title found for URL {response.url}"
                    )
        except Exception as e:
            custom_log_error(
```

```
                f"Error extracting product title for product: {response.
                    url}: {e}"
            )
        # Extracting other product details.
        # Code ommited.

    product_detail_item = loader.load_item()
    yield product_detail_item
```

In this method, our custom item processing pipeline processors.py is used to aggregate and standardize the extracted data. The loader applies input and output processors like MapCompose and clean_description to ensure data consistency and cleanliness.

The spider's completion and performance are logged in the closed method, which records the total number of products scraped and the duration of the scraping process.

```
# Code Snippet
def closed(self, reason):
    # Clear the list for the subcategory after processing
    self.subcategory_product_links = {}
    end_time = time.time()  # Record the end time
    duration = end_time - self.start_time  # Calculate the duration

    # Convert seconds into hours, minutes, and seconds
    hours, remainder = divmod(duration, 3600)
    minutes, seconds = divmod(remainder, 60)

    # Format the time into a readable string
    formatted_duration = (
        f"{int(hours)} hours : {int(minutes)} minutes : {int(seconds)} "
            seconds"
    )

    # code ommited
    custom_print(f"Total products scraped: {self.product_count}")
    custom_print(f"Total time taken: {formatted_duration}")
    custom_print(f"Spider closed: {reason}")
```

4.3.2 Scraper Implementation for venekauppa.com (venspider)

Website Investigation and Planning

We followed the same method that we used for bikmarine.fi. First we checked its robots.txt file.

```
User-agent: *
Host: venekauppa.com
Sitemap: https://venekauppa.com/sitemap.xml
```

Figure 9: robots.txt from venekauppa.com

Based on the Figure 9, it also does not disallow scraping product data from the website. It also provides the location of the site's sitemap (Sitemap: https://venekauppa.com/sitemap.xml), which provides a roadmap of the site's structure for more efficient indexing.

Website Structure

The venekauppa.com website also does not use JavaScript rendering to show information on the website. It makes it alot easier to scrap the data from the website. We can use Scrapy CSS and xpath selectors to get the data.

We first checked the location of product links in HTML, as shown in Figure 10.

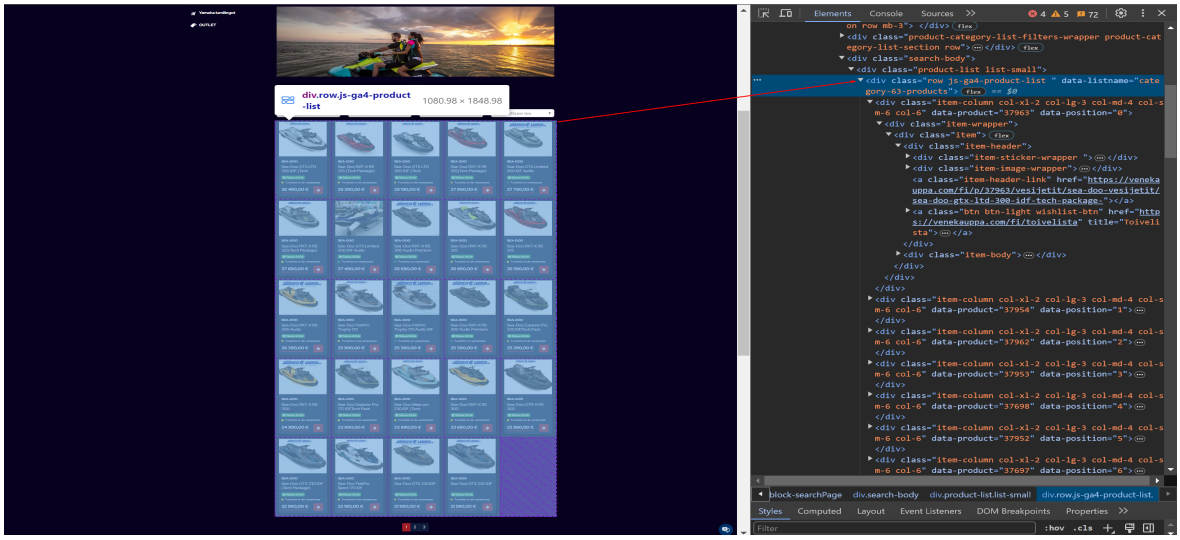

Figure 10: Location of Product Links on venekauppa.com

All the product in a subcategory are in a tag. Each product is represented by a tag within tag. Each tag within tag represnts information about specific product, including its links which is in <a href""> tag. We will use this information to build our selectors to get product links.

venekauppa.com also has multiple pages in each subcategory that includes more products. From Figure 11, we can see that these pages are in a <ul> tag and each <li> tag inside the

 tag includes information about the current page and also the links to the next page. We also need to make sure that we get the links for the next pages and handle them to get all product links from every page for a specific subcategory.

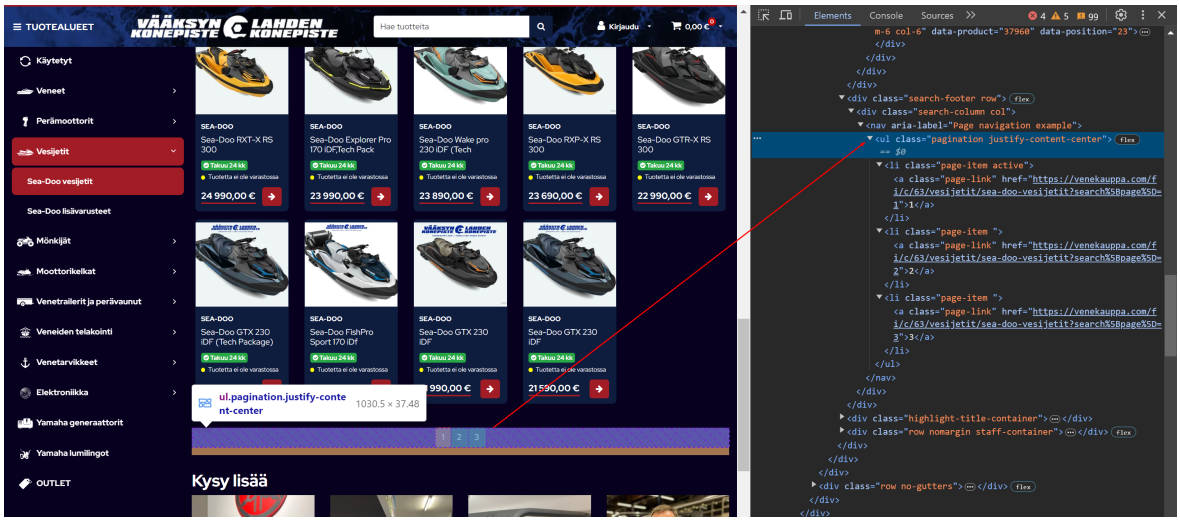

Figure 11: Pagination on venekauppa.com

We then move to getting product details for each product. For this we will investigate each product page.

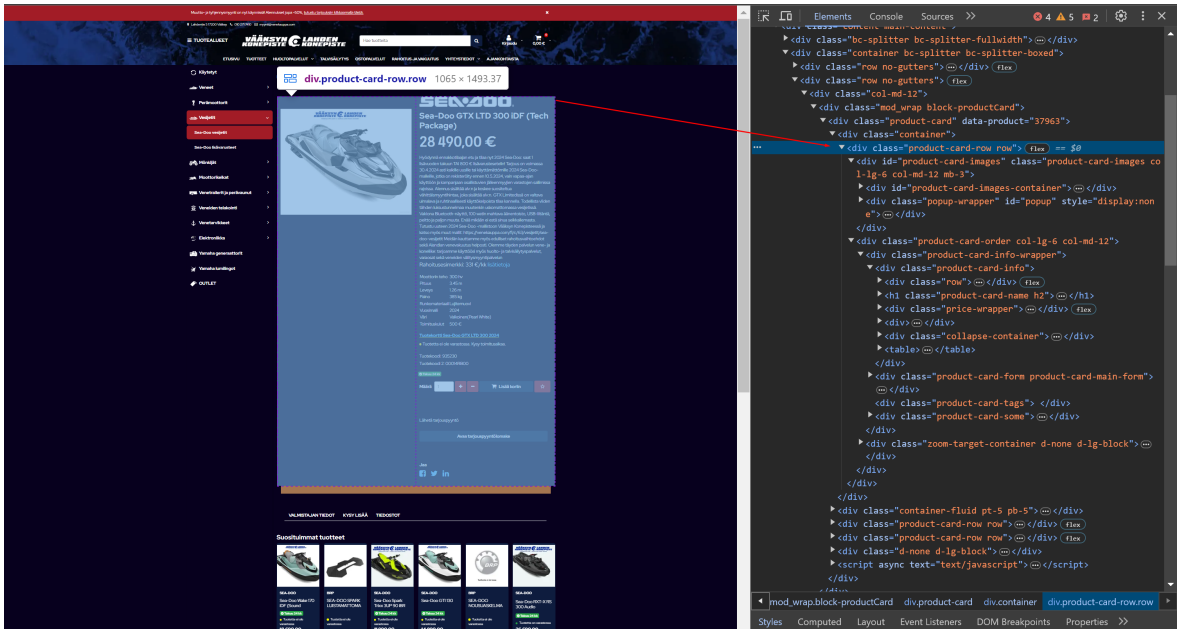

Figure 12: Location of Product Details on venekauppa.com

From Figure 12, we can see that all information about the product, title, images, description, price, etc. are inside <div class="“product-card”> tag. We further investigated this to get specific location of each data point required for our project.

Scraper Implementation

After instigating the website and locating our data points, we defined our css and xpath selectors in website-config file. These selectors as shown in Figure 13, will be used by the spider for venekauppa.com.

```
# A dictionary of CSS selectors that are used to extract specific data from the website.
"selectors": {
    "product_links": "div.item-wrapper a.item-header-link::attr(href)",
    # "product_code": "div.form-row > div.product-card-form-additional-wrapper > span.product-card-form
    "title": ".product-card-info .product-card-name.h2::text",
    "product_images": "div.product-card-slider div.item picture img::attr(src)",
    "brand": [
        "div.col-12 a picture img::attr(alt)",
        "span.product-card-manufacturer::text",
    ],
    "model": '//tr[td[text()="Vuosimalli"]]/td[2]/text()',
    "description": [
        "div#tech::text, div#tech p::text",
        '//div[@id="basic"]/text()',
    ],
    "monthly_price": "span.product-card-toggle-area-title.price.financing-price::text",
    "leasing_price": '//h2[@class="product-card-price display-4 custom-font"]/text()',
    "number_of_payments": "span.font-weight-bold::text",
    "pagination": "ul.pagination li.page-item a.page-link::text",
    # ... other CSS selectors specific to venekauppa
},
```

Figure 13: CSS and Xpath Selectors for venekauppa.com

Data Collection Process

The scraper design is the same as we implemented for bikemarine.fi. We first initiates the data collection process with the start_requests method, which selects the venekauppa.com website from a predefined list of categories and then calls the venekauppa_requests function to start scraping.

The only difference is that we are also doing pagination for venekauppa.com for each sub-category with multiple pages and getting the links.

```
# Code Snippet
# Check for pagination and handle it
base_url = response.url.split("?")[0]
last_page_elements = response.css(selectors["pagination"]).getall()
if last_page_elements:
    last_page = int(last_page_elements[-1])
    if "search%5Bpage%5D=" in response.url:
        current_page = int(response.url.split("search%5Bpage%5D=")[-1])
    else:
        current_page = 1

    # If there are more pages, schedule requests for them
    if current_page < last_page:
```

```
next_page = current_page + 1
next_page_url = f"{base_url}?search%5Bpage%5D={next_page}"
yield scrapy.Request(
    next_page_url,
    callback=self.venekauppa_parse_products_links,
    meta={
        "subcategory": subcategory,
        "category_name": category_name,
        "website": website,
    },
)
return
```

The venekauppa_requests method sends HTTP requests to each subcategory within the venekauppa.com website. It uses Scrapy's Request function to asynchronously fetch the page content.

Upon receiving a response, the venekauppa_parse_products_links method parses the HTML content to extract product links using CSS selectors. These selectors are configured specifically for the venekauppa.com website.

Data Processing and Loading

The data processing and loading implementation for venekauppa.com is also the same as we did for bikemarine.fi.

For each product link, the venekauppa_parse_product_details method is invoked. This method is the core of the data extraction process, where detailed information about each product is obtained. The custom item processing pipeline processors.py is used to aggregate and standardize the extracted data.

The spider's completion and performance are logged in the closed method, which records the total number of products scraped and the duration of the scraping process.

4.3.3 Scraper Implementation for tietokonekauppa.fi (tietospider)

Website Investigation and Planning

We followed the same method as used for bikmarine.fi and for vennekaupa.com. First we checked its robots.txt file.

```
User-agent: AhrefsBot
Disallow: /

User-agent: MJ12bot
Disallow: /

User-agent: ltx71
Disallow: /

User-agent: Tyxobot
Disallow: /

User-agent: Riddlerbot
Disallow: /

User-agent: wget
Disallow: /

User-agent: *
Disallow: /products/find/
Disallow: /products/show/
Disallow: /cart/

Sitemap: https://tietokonekauppa.fi/sitemap.xml
Sitemap: https://tietokonekauppa.fi/sitemap_1.xml
Sitemap: https://tietokonekauppa.fi/sitemap_2.xml
```

Figure 14: robots.txt from tietokonekauppa.fi

Based on the Figure 14, It blocks certain crawlers (like AhrefsBot, MJ12bot, ltx71, Tyxobot, Riddlerbot, wget) from accessing any part of the site (Disallow: /). For all other crawlers (User-agent: *), it restricts them from accessing specific directories related to product search and cart functionalities (Disallow: /products/find/, /products/show/, /cart/). As we will not be targeting these specific path mention in the robots.txt file, we should be good. Also as mention before nettileasing has a contract in place with these websites to scrap products data.

Website Structure

While the product information on the tietokonekauppa.fi does not use JavaScript rendering, it uses sliding pagination rendered by javascript as shown in Figure 15. This means that when we move from page to page in each subcategory the page numbering changes based on which page we are currently viewing. tietoSpider integrates Scrapy with Playwright to handle dynamic pagination loaded via JavaScript.

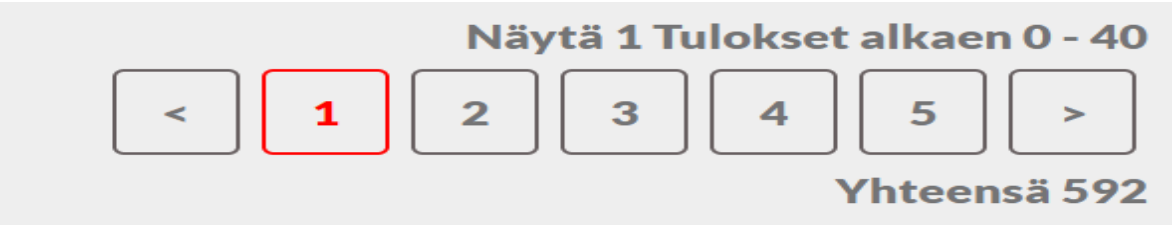

Figure 15: tietokonekauppa.fi Sliding Pagination

We first checked the location of the product links, then we checked the location of each product details.

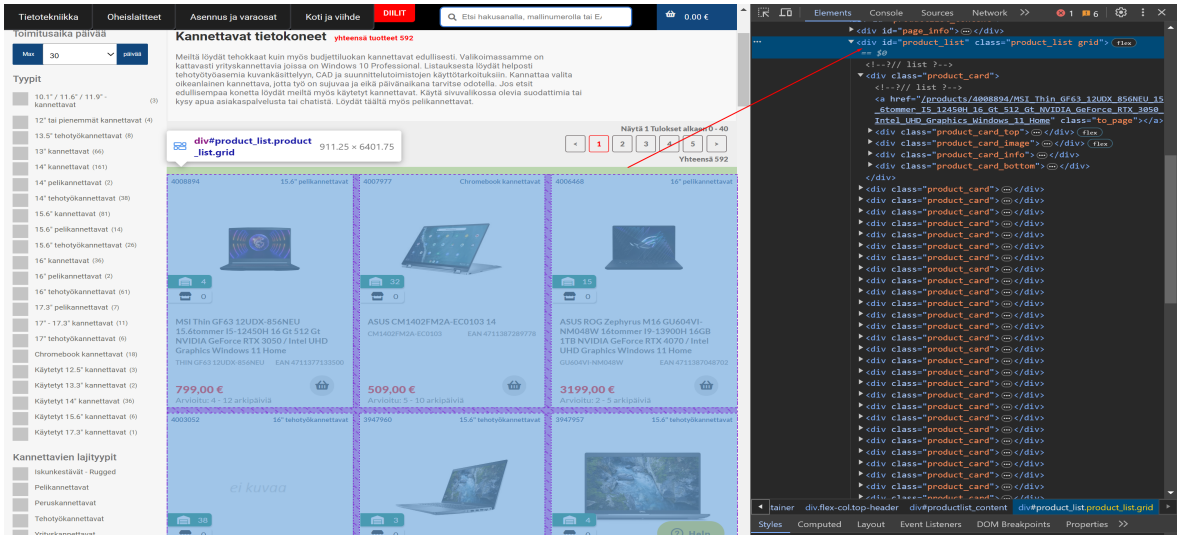

Figure 16: Location of Product Links on tietokonekauppa.fi

We can see in Figure 16, that all the product in a subcategory are in a tag. Each product is represented by a tag within tag. Each tag within tag represents information about specific product, including its links which is in <a href“”> tag. We will use this information to build our selectors to get product links. tietokonekauppa.com also has multiple pages in each subcategory that includes more products. As these page numbers are dynamically loaded, we will use playwright to handle pagination. We then move to getting product details for each product. For this we will investigate each product page.

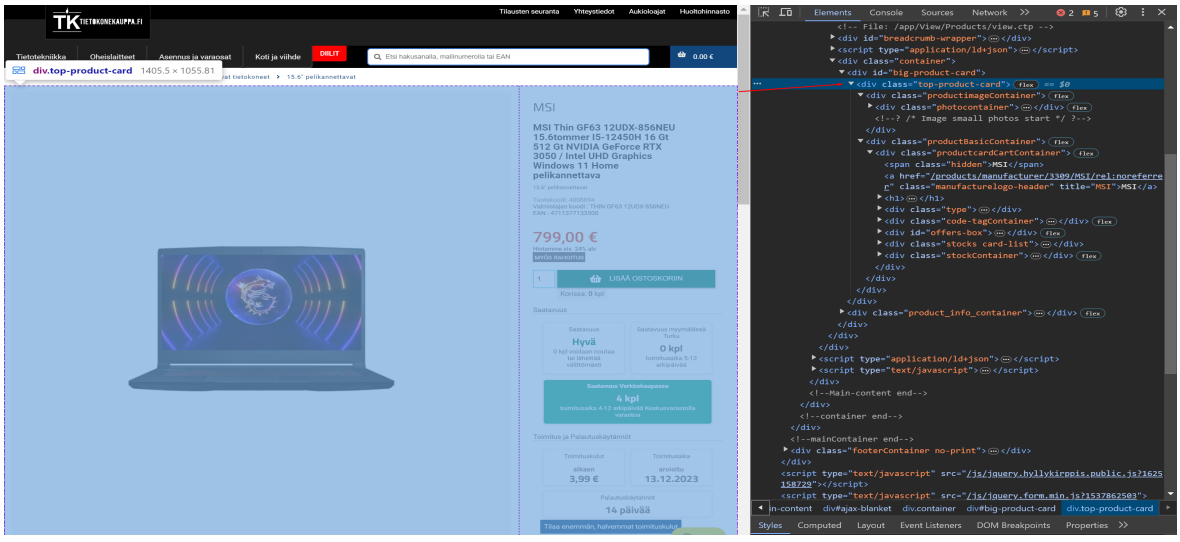

Figure 17: Location of Product Details on tietokonekauppa.fi

From Figure 17, we can see that all information about the product, title, images, description,

price, etc. are inside tag. We further investigated this to get specific location of each data point required for our project.

Scraper Implementation

After instigating the website and locating our data points, we defined our css selectors in website-config file. These selectors as shown in Figure 18, will be used by the spider for tietokonekauppa.fi.

```
# A dictionary of CSS selectors that are used to extract specific data from the website.
"selectors": {
    "product_links": "#product_list .product_card .to_page::attr(href)",
    "title": ".productcardCartContainer > h1::text",
    "product_code": "div.ourcode::text",
    "product_images": ".photocontainer .mainPhoto a::attr(href)",
    "brand": ".manufacturelogo-header::text",
    "model": "",
    "description": ".descriptionContainer",
    "monthly_price": ".pricebox .smalltext .redtext::text",
    "leasing_price": ".part_ask .redtext::text",
    "number_of_payments": ".pricebox .smalltext .redtext::text",
    # ... other CSS selectors specific to tietokonekauppa
},
```

Figure 18: CSS Selectors for tietokonekauppa.fi

Data Collection Process

The scraper design is the same as we implemented for bikemarine.fi and venekauppa.com. We first initiates the data collection process with the start_requests method and then calls the tietokonekauppa_requests function to start scraping.

The tietokonekauppa_subcategory_request and extract_all_product_links methods handles the response, leveraging Playwright to navigate through the pages and extract product links. It also manages the browser instance for efficient navigation and data extraction.

```
# Code Snippet
def tietokonekauppa_subcategory_request(self, response):
    if not self.is_browser_open():
        self.launch_browser()
    self.page = self.browser.new_page()
    all_links = self.extract_all_product_links(response.url)
    for product_link in all_links:
        yield scrapy.Request(
            product_link,
            callback=self.tieto_parse_product_details,
            meta={"subcategory": response.meta["subcategory"], "
                category_name": response.meta["category_name"], "website":
```

```
                response.meta["website"]}
        )

# Code Snippet
# Handeling pagination with playwright
def extract_all_product_links(self, initial_url):
    # Initialize a set to store unique product links.
    product_links = set()

    # Navigate to the initial URL using the browser automation tool.
    self.page.goto(initial_url)

    # Wait for the webpage's content to be fully loaded.
    self.page.wait_for_load_state("domcontentloaded")

    # Wait for a specific element that indicates the page has loaded.
    self.page.wait_for_selector(".product_list")

    # Initialize the current page number.
    current_page = 1

    # Loop until all product links are extracted from all pages.
    while True:
        # Extract product links from the current page.
        # We are targeting product links with specific CSS selectors.
        current_links = self.page.eval_on_selector_all(
            "#product_list .product_card .to_page",
            "nodes => nodes.map(n => n.href)",  # Map each node to its
                href attribute to get the link.
        )

        # Update our set with newly extracted product links.
        product_links.update(current_links)

        # Look for the 'next page' button. If it's not found or inactive,
            it means we're on the last page.
        next_page_button = self.page.query_selector(
            f'.pagination .page[data-page="{current_page + 1}"]:not(.
                active)'
        )

        # If no more pages are available, break out of the loop.
        if not next_page_button:
            custom_print("All pages Extracted.")
            break
```

```
            # Otherwise, click the 'next page' button to load the next page.
            next_page_button.click()

            # Again, wait for the content to load before extracting links.
            self.page.wait_for_load_state("domcontentloaded")

            # Wait for a specific element that indicates the page has loaded.
            self.page.wait_for_selector(".product_list")

            # Increment the page number.
            current_page += 1

    # Return the set of all extracted product links.
    return product_links
```

Data Processing and Loading

The data processing and loading implementation for tietokonekauppa.fi is also the same.

For each product link, the tieto_parse_product_details method is called. This method is the core of the data extraction process, where detailed information about each product is obtained. The custom item processing pipeline processors.py is used to aggregate and standardize the extracted data.

The spider's completion and performance are logged in the closed method, which records the total number of products scraped and the duration of the scraping process.

4.4 Customizing the User Interface

4.4.1 GUI Design

The design and functionality of the Graphical User Interface (GUI) shown in Figure 19 for the Nettileasing web scraping system play a pivotal role in controlling and monitoring the scraping process for each website. Developed using Tkinter in Python, the GUI offers a user-friendly interface, providing real-time control and feedback for the scraping tasks. The GUI is a standalone feature for the system, we built it with modularity in mind. We can remove any specific spider without affecting the GUI or we can remove the GUI in case we want to deploy the scraper on a cloud service, it will not affect any of the system functionality.

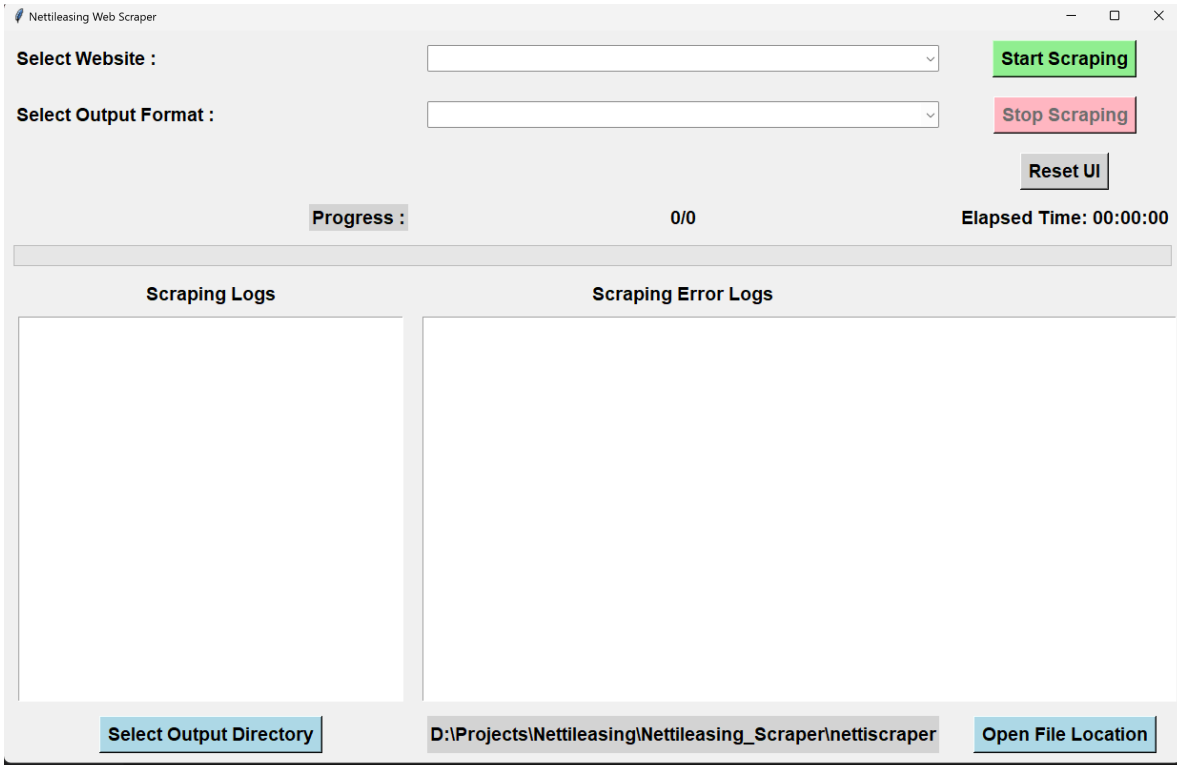

Figure 19: Nettiscraper GUI

4.4.2 Layout and Aesthetics

As shown in Figure 19, the GUI layout is structured for simplicity and ease of use. It includes dropdown menus for website and output format selection, buttons for starting and stopping the scraping process, and a dedicated area for displaying logs and progress.

Website Selection Dropdown: A dropdown menu allows the user to select the target website (Bike Marine, Venekauppa, or Tietokonekauppa) or choose "All" for scraping all websites sequentially.

Output Format Selection: Another dropdown menu enables the selection of the desired output format (CSV or JSON), catering to different data utilization needs.

Control Buttons: The interface includes Start Scraping, Stop Scraping, and Reset UI buttons, providing users complete control over the scraping process.

Real-time Log Display: A text area is dedicated to displaying real-time logs, offering transparency and immediate feedback on the scraping process.

Progress Monitoring: A progress bar and time elapsed label are included, providing visual feedback on the scraping progress and duration.

Directory Selection: The interface allows users to select an output directory for the scraped data, making data management more flexible.

4.4.3 Testing Strategies

Testing the GUI involved a combination of manual and automated testing methodologies:

Manual Testing: The GUI was rigorously tested for user interaction, ensuring that all elements respond as expected. This included testing dropdown selections, button functionalities, and the overall user experience.

Integration Testing with Scrapers: The GUI was tested in an integrated environment with the scrapers to ensure seamless interaction between the GUI and the scraping processes.

Performance Testing: The GUI's performance, particularly in terms of response time and resource usage was monitored to ensure that it remains efficient and responsive even during extensive scraping tasks.

4.4.4 Performance Optimization

Optimizing the GUI for performance was crucial, especially given the resource-intensive nature of web scraping.

Efficient Event Handling: Event handling within the GUI, particularly for start, stop, and reset actions, was optimized to prevent any lag or unresponsiveness.

Background Processing: Scraping tasks are executed in background threads, ensuring that the GUI remains responsive during the scraping process.

```
# Code Snippet
self.crawler_thread = threading.Thread(target=self.run_spider, args=(
    selected_site, output_format))
```

Resource Management: The GUI efficiently manages system resources, avoiding excessive CPU or memory usage, which is particularly important during long scraping sessions.

Error Handling in the GUI: Robust error handling within the GUI ensures that any issues during scraping are gracefully managed and communicated to the user through the log display.

The custom GUI for the Nettileasing web scraping system is designed to be intuitive, responsive, and efficient, providing users with comprehensive control and monitoring capabilities.

Its seamless integration with the scraping system ensures a smooth and effective scraping operation. The GUI is designed to be responsive, ensuring usability across different screen sizes and resolutions.

4.5 Challenges during Implementation

When the project was started, the initial plan was to develop the scraping system and deploy it on amazon AWS. Nettileasing already has a website which is deployed on aws, so our initial plan was to deploy the scraper on aws and develop some sort of API or plugin for thier existing website which will automatically fetch the scraped data from our scraper and show it on their website. But, as we started the development process and started to dig deep we faced a few problems:

4.5.1 Existing Website

The first issue we encountered was that the developer that worked on the development of nettleasing's website left the company and there was no documentation on how the website was developed or how to setup a development environment for it so that we add more features. The lack of documentation on the website was the biggest hurdle. We tried reaching out to the developer, but the response we got was not helpful. The lack of access to their website's database and to the code made it impossible to make changes to it.

4.5.2 Budget Constraints

As nettileasing is a startup, there were budget constraints. Their website was deployed on AWS, deploying a large scale scraper will add to the cost. The company did not want to add more cost until they start making income from their website.

4.5.3 Time Constraints

Our timeline was 3 months for the whole scraping system and thesis writing. It is too short to develop a large scale scraping solution and also integrate it with their current website, which had no documentation. To develop the integration part, we would need to learn all the technologies used in their current website and also understand their code which was not possible in this short time.

4.5.4 Solution

After careful consideration and meetings with our client at Nettileasing Finland Oy, it was agreed upon that initially we will develop a desktop based web scraping solution. This gives them a high level of control over the scraping process and they can get the scraped data and

upload the file into their website. But the solution should be modular, so that in future we can easily deploy it on the cloud. The developed system was then modularized, so that we can easily deploy it on the cloud without the need for making major changes the code, this included a standalone GUI which can be removed without affecting the scrapers for each website. The former developer of nettileasing's website agreed to develop a module for their website where we can bulk upload product data from a csv or json file.

4.6 Implementation Process and Testing

4.6.1 Development Workflow

The implementation of each scraper for Nettileasing Finland Oy's web scraping system followed a structured development workflow, ensuring that each step contributed to a robust and effective final product. Weekly meeting were arranged with our clients at nettileasing to ensure that the project was going in the right direction and fulfill their requirements.

Requirement Analysis: Initially, a thorough analysis of each target website (bikemarine.fi, venekauppa.com, tietokonekauppa.fi) was conducted. This involved understanding the website's structure, the nature of the data to be extracted, and any specific challenges each site presented.

Design and Planning: Based on the analysis, a design for each scraper was outlined. This included selecting the appropriate tools (Scrapy, Playwright) and defining the data models in items.py.

Implementation: The spiders were then implemented, with each website requiring tailored solutions. For instance, tietokonekauppa required Playwright integration for dynamic content handling, whereas bikemarine and vennekaupa primarily utilized Scrapy's capabilities. As we developed each prototype, it was shown to the clients at nettileasing and changes were made according to their feedback.

Code Refinement and Modularization: The code was continuously refined and commented for clarity and efficiency. Common functionalities, such as error handling and data processing, were modularized in middlewares.py, pipelines.py, and processors.py.

Meetings with Client: We conducted regular weekly meeting with our client at Nettileasing Finland Oy. These meeting were conducted to show the prototype that were developed and any challenges faced during the development process. This ensured that the scraping system was according to their expectation. Detailed weekly reports were also provided to nettileasing which included the status of the project and planning for the next development

phase.

Documentation: Every phase of the development was documented. This involved the results from the prototype that was developed and compared to the nettileasing existing manual methods.

4.6.2 Testing Strategies

For each scraper, a comprehensive testing strategy was employed, incorporating both unit and integration tests.

Functional Testing: Each spider was tested in isolation to ensure it correctly navigated the target sites, extracted the required data, and handled errors gracefully.

Unit Tests: Focused on testing individual components (like functions in processors.py) in isolation. This included testing the data processing logic and custom functions for accuracy and reliability.

```
# Code Snippet
# Example unit test for a data processing function
def test_clean_price():
    assert clean_price('$99.99') == '99.99'
```

Integration Tests: These tests evaluated how well the different components of each spider worked together. They included testing the entire scraping process from making requests to processing and storing data.

Integration tests were particularly important for tietokonekauppa due to the integration of Playwright.

Website-Specific Testing: Each spider was tested against the specific challenges and structures of its target website. For instance, tests for tietokonekauppa included scenarios where dynamic content was loaded after user interactions.

4.6.3 Performance Optimization

Performance optimization was a key focus throughout the development process:

Request Management: Efficient management of HTTP requests was crucial. Techniques like request throttling, concurrent requests, and proper use of Scrapy's settings (DOWNLOAD_DELAY, CONCURRENT_REQUESTS) were employed to balance speed and server load.

Resource Utilization: For tietokonekauppa, resource-intensive operations like browser automation were optimized by managing the Playwright instances effectively, ensuring they were closed after each scraping task to free up resources.

Data Processing Optimization: The data processing pipelines were optimized for speed and efficiency. This included streamlining the data cleaning and validation processes to reduce processing time.

Memory and CPU Usage: Memory and CPU usage were monitored and optimized. This included fine-tuning the spider's settings and code to ensure they were resource-efficient.

In summary, the implementation process and testing of the web scraping system for Nettileasing Finland Oy were carried out with a focus on thoroughness, efficiency, and performance optimization. Each scraper was meticulously tested and refined to ensure it met the operational requirements, and the system as a whole was optimized to function effectively while minimizing resource usage.

5 Results

5.1 Performance Evaluation of the Web Scraping System

Web Scraping Performance

The performance evaluation of the web scraping system is important for understanding its efficiency and effectiveness. In addition to the website structure, static or javascript rendering, the scraping performance is also influenced by the setting DOWNLOAD_DELAY and the number of CONCURRENT_REQUESTS.

A high download delay helps to avoid IP bans and overloading of the target websites servers, but it significantly lowers scraping performance. Setting concurrent requests to a higher number allows the scraper to process multiple items at the same time, contributing to the overall efficiency of the system, as seen by Venekauppa's high products per minute rate. The variations in products per minute across websites can be attributed to these settings, combined with the inherent complexities of each website's structure and data availability.

Website	**Total Products Scraped**	**Total Time Taken**	**Products per Minute**
Bike Marine	57	39 seconds	87.69
Venekauppa	761	6 minutes 40 seconds	114.15
Tietokonekauppa	14,608	6 hours 15 minutes 59 seconds	38.97

Table 7: Web Scraping Performance

The scraping performance as shown in Table 7, of each site varied significantly, reflecting the distinct nuances of each web site. Bike Marine showed an efficient scraping approach for smaller-scale data extraction, as seen by the 87.69 rate of products per minute and the 39 second duration. Venekauppa, despite having a moderate scraping time of 6 minute and 40 seconds, demonstrated an impressive 114.15 rate of product extraction per minute, showcasing the system's ability to adapt to different website structures effectively. Tietokonekauppa, with the largest volume of product data of 14,608 and the longest scraping duration 6 hours 15 minutes and 59 seconds, showed a lower 38.97 rate of products per minute, which is expected given the complexity and volume of data, website structure and javascript rendering involved.

Technical Performance

Table 8, shows the metrics for assessing the technical performance of the system.

Metric	Value	Remarks
Average Products Extracted per Minute	40.25	Efficient extraction of a high volume of products, showcasing the system's robust processing ability.
Accuracy (per 1000 products) (%)	99.99	High level of precision in data extraction, indicating accurate scraping.
Error Rate (per 1000 products) (%)	0.1	Low error rate, acceptable for complex scraping operations.
System Stability (up-time %)	100	High stability, ensuring consistent data collection with no downtime.
CPU Load (%)	37.00	Efficient resource utilization on multi-core processor, not overburdening the processor.
Memory Usage (GB)	2.26	Moderate usage, within expected range for web scraping systems.

Table 8: Technical Performance

The capacity of the web scraping system to extract an average of 40.25 products per minute demonstrates its ability for handling large amounts of data efficiently. This data extraction rate is especially important in real-world e-commerce settings where system throughput is an important indicator of performance. The system's impressive 99.99% accuracy rate shows the reliability of the collected data, which is essential for reducing errors and guaranteeing the integrity and quality of the information collected. This high level of accuracy is noteworthy, considering the inherent challenges posed by the diverse and dynamic nature of web pages encountered during scraping.

The system's low error rate of 0.1% is equally important. This figure indicates the system's robustness and ability to navigate the complexities associated with scraping a variety of web environments. Moreover, the system's 100% up-time highlights its reliability and consistent operating capability, which is an important aspect for keeping up-to-date product listings. In the fast-paced world of e-commerce where timely data availability may greatly affect corporate strategy, this continuous operating capacity is essential.

The system performs well with minimal CPU and memory consumption. Such resource utilization efficiency is essential for sustaining large-scale scraping operations without overburdening the underlying hardware infrastructure. Together, these metrics show that the web scraping system is not only very accurate and efficient, but also reliable and resource-conscious. Its optimal performance combined with efficient resource management makes it a useful tool for large-scale data extraction.

5.2 Comparative Analysis with Manual Data Collection Methods

The comparative analysis of the automated web scraping system against manual data collection methods reveals significant insights into the efficiency and effectiveness of automated processes. The real-world outcomes of scraping tasks on sites like Tietokonekauppa, Venekauppa, and Bike Marine give a solid foundation for this comparison.

Efficiency of Automated Scraping

The scraping process for Tietokonekauppa, which included obtaining a significant amount of data (14,608 products), took roughly 6 hours and 16 minutes. This level of efficiency stands in stark contrast to what might be accomplished manually. To put this into context, manual data collecting methods would take substantially longer. It will take several weeks or months to acquire the same volume of data, due to the time required to navigate each page, copy information, and ensure accuracy.

Time-Saving Potential

Automated scraping demonstrates immense time-saving potential, especially in complex e-commerce environments like Tietokonekauppa. The large inventory and complex web structure makes manual data collection a highly time-intensive task. The ability of the scraper to process high volumes of data at a rate of 38.97 products per minute, as seen in Tietokonekauppa's case, highlights the system's capability to handle large-scale and complex scraping tasks efficiently.

Adaptability and Scalability

The diverse nature of the scraped websites (Bike Marine, Venekauppa, Tietokonekauppa) and the scraping system's adaptability to each setting shows its scalability. An advantage of the system over manual techniques is its capacity to modify its scraping strategy in response to changes in the website's structure and data availability.

Practical Implications

In practical terms, the efficiency and time-saving aspects of the automated web scraping system translate into significant cost savings and resource optimization. Manual data collection not only requires more time but also incurs higher costs in terms of labor and potential errors. The ability to rapidly collect and update data ensures that businesses can make timely decisions based on current market trends and consumer preferences, which is crucial in dynamic industries like e-commerce.

Manual data collection is inherently limited in several ways. It lacks the scalability needed for processing large volumes of data or expanding to multiple data sources which is effortlessly managed by the automated system. Additionally, manual methods fall short in providing real-time data access, an increasingly important feature in dynamic sectors like e-commerce. Another advantage of automated system over manual method is that it offers up-to-date information. Furthermore, manual data collection demands significant human resource allocation, which could be more strategically employed if an automated system is in place.

5.3 Impact on Business Operations and Customer Retention

Operational Efficiency

Operations were greatly improved for Nettileasing Finland Oy with the introduction of an automated web scraping technology. The goal of the system's design was to automate the process of obtaining product data from pre-selected websites, which included prices, images, descriptions, and other relevant details. The previous methods were replaced by this automation, which increased the speed and efficiency of data gathering. The ability to handle dynamic material and manage data extraction from different sources efficiently handled technical problems while ensuring scalability and reliability. Nettileasing Finland Oy saw an increase in operational efficiency and more timely and accurate data collection by integrating this user-friendly solution into their current website. This is important because the e-commerce industry is changing quickly. This rapid and accurate data collection is important in evolving e-commerce sector.

Customer Insights

The solution delivered more accurate and timely data, which resulted in better consumer insights and decision-making. Nettileasing Finland Oy can now offer customers with up-to-date information by utilizing the web scraping system's ability to update data in real-time, hence improving the user experience on their platform. This also allowed for more informed company decisions and strategy development, which aligned with the company's aim of maximizing data-driven e-commerce strategies.

Customer Retention

Although we could not directly link improved customer retention to our system due to lack of data, the improved user experience that resulted from the adoption of the web scraping technology might improve customer satisfaction and retention. The system's data collection accuracy and timeliness increased the platform's reliability and usefulness for Nettileasing

Finland Oy. This reliability and user-friendly nature of the platform likely increased customer loyalty and engagement, as users could trust the platform for up-to-date and accurate information. The improved customer experience was a direct outcome of the web scraping system's efficiency and accuracy enhancements.

Revenue Growth

While the thesis does not address direct revenue growth, it is possible to assume that greater operational efficiency, enhanced customer insights, and increased customer retention may have contributed to revenue growth. Nettileasing Finland Oy gained a competitive advantage through the automation of data collection and the resulting accurate and timely information. In a market where real-time data and user trust are critical, this competitive advantage most certainly translated into increased user traffic and maybe higher conversion rates, significantly boosting the company's income.

The automated web scraping system's adoption had a considerable positive influence on Nettileasing Finland Oy's company operations and customer retention. The solution increased operational efficiency, gave deeper insights into customers, provided better user experience, and probably led to customer loyalty and revenue growth. These improvements are in line with the thesis's objectives and research questions, demonstrating the practical implications of using an automated web scraping system in an e-commerce setting.

6 Discussion

This thesis has explored automated web scraping technologies in e-commerce. Our main focus has been on how these technologies affect data collection, making it more efficient and accurate. In this discussion, we connect our findings to existing research and industry practices. We look at how technological advancements not only improve data collection methods but also how they might change the way customers are engaged and kept. Our case study of Nettileasing Oy Finland provides a real-world example. We will revisit our research questions to better understand our results. We aim to show what our findings mean for the future of technology in e-commerce.

6.1 Results in Relation to Research Questions

Q1: How does the implementation of an automated web scraping system improve the accuracy and efficiency of e-commerce products data collection compared to manual methods for Nettileasing Finland Oy?

The findings of this study clearly show that using an automated web scraping technology improves the accuracy and efficiency of data collecting in e-commerce. This is consistent with the previous research (Jordan & T. M. Mitchell, 2015), which highlights the transformational power of technology in data management processes. As demonstrated by its implementation for Nettileasing Finland Oy, the ability of the system to handle massive and dynamic e-commerce data shows a significant improvement over conventional manual approaches. This study matches with academic and practical perspectives on the need of technical innovation in managing huge amounts of unstructured data, which is typical in e-commerce contexts.

Q2: Does improving accuracy and efficiency of data collection lead to improved customer satisfaction and retention?

While the study also aimed to explore the correlation between improved data collection and customer satisfaction, the available data was insufficient to conclusively demonstrate this relationship. However, existing literature like in Jorge et al., 2020, supports the notion that enhanced data accuracy and efficiency can positively impact customer satisfaction in e-commerce. This suggests that while the study could not empirically establish this correlation, the theoretical framework and prior research supports the idea that efficient and accurate data collection and implementation may have a major impact on customer satisfaction in e-commerce, implying that such systems have the potential to improve customer relationship management.

6.2 Explanation and Interpretation of Results

The effectiveness of the web scraping system in improving data accuracy and efficiency can largely be attributed to the advanced tools and algorithms used. These tools and algorithms, designed for high performance and precision, significantly reduced errors compared to manual methods. E-commerce data is dynamic, vast, and often unstructured. The web scraping algorithm excelled at handling this complexity, resulting in more accurate and efficient data collection.

Another factor was the unique requirements and context of Nettileasing Finland Oy. The web scraping system was customized to meet the company's specific data requirements, ensuring that the system was not only effective in general but also particularly efficient in the context of Nettileasing Oy's operations. These elements all contributed to the findings in our study, showing the immense potential of automated web scraping in enhancing data gathering processes in the e-commerce business.

The outcomes of this study provide a significant contribution to the fields of web scraping technology and e-commerce. Businesses aiming to improve their data management strategies might gain useful insights from the observed improvements in data collection made possible by the automation process. This study emphasizes the significance of advanced data processing tools in improving customer experiences and potentially improving customer retention, implying a compelling reason for companies to invest in such tools. Furthermore, it establishes a framework for future study into the use of web scraping technologies in data-driven business strategies.

7 Conclusion and Recommendations

As this thesis concludes, it is important to reflect on the comprehensive journey that began with a thorough investigation of the significance of automated web scraping technologies versus human data gathering techniques in e-commerce. The research provided an in-depth understanding of the intricacies and complexity involved in data collection in the dynamic world of e-commerce.

The extensive analysis of the study led to a number of significant findings which included a performance evaluation of the web scraping system, a comparison with manual data gathering techniques, and an assessment of its impact on business operations and possible customer retention. The findings shows that automated web scraping techniques are not only operationally superior than manual approaches, but also has strategic ramifications for e-commerce businesses. As demonstrated in the case studies of Nettileasing Finland Oy, the efficiency, accuracy, adaptability, and scalability of automated systems highlight a fundamental change in e-commerce toward advanced data-driven strategies.

Looking back at this research from a broader perspective, it is clear that using automated web scraping technologies has implications that go beyond just improving operational efficiency. They touch on aspects of strategic market positioning, customer engagement, and long-term business sustainability. Integration of such technologies is an important step for companies hoping to survive in the increasingly competitive and data-driven world of e-commerce.

In the future, the study provides opportunities for further research in a number of important areas. The potential integration of AI and machine learning with web scraping tools presents an exciting opportunity for even more sophisticated data analysis and adaptive scraping strategies. Furthermore, as the digital world evolves, there is an increasing need to explore the ethical and legal components of web scraping in order to guarantee responsible and compliant data gathering techniques.

In conclusion, this study has offered useful insights into the benefits of automated web scraping in e-commerce. It addressed the research question while emphasizing the importance of technology-driven approaches to data collecting and analysis. These findings have implications that go beyond the academic sphere and provides practical direction for e-commerce businesses. Future study in this field should help us better understand and apply web scraping technologies in the rapidly evolving digital ecosystem.

References

APIFY (2023). *JSDOM Scraper · Apify*. [Online; accessed 13. Oct. 2023]. URL: https://apify.com/apify/jsdom-scraper.

Brin, S. & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* 30(1), pp. 107–117. ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X.

Chaffey, D. & Ellis-Chadwick, F. (2019). *Digital marketing*. Seventh edition. Harlow, England ; Pearson. ISBN: 978-1-292-24157-9.

Chaulagain, R. S., Pandey, S., Basnet, S. R. & Shakya, S. (2017). Cloud Based Web Scraping for Big Data Applications. In: *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 138–143. DOI: 10.1109/SmartCloud.2017.28.

Chen, D. Q., Preston, D. S. & Swink, M. (2015). How the Use of Big Data Analytics Affects Value Creation in Supply Chain Management. *Journal of Management Information Systems* 32(4), pp. 4–39. DOI: 10.1080/07421222.2015.1138364.

Dalessio, M. (Sept. 25, 2023). *Nokogiri*. [Online; accessed 13. Oct. 2023]. URL: https://nokogiri.org/index.html.

Diouf, R., Sarr, E. N., Sall, O., Birregah, B., Bousso, M. & Mbaye, S. N. (2019). Web Scraping: State-of-the-Art and Areas of Application. In: *2019 IEEE International Conference on Big Data (Big Data)*, pp. 6040–6042. DOI: 10.1109/BigData47090.2019.9005594.

Divya Khairkar, N. R. (2023). E-Commerce Product Price Tracker using Dynamic Pricing Algorithm. *International Journal for Research in Applied Science and Engineering Technology* 11. DOI: 10.22214/ijraset.2023.51478.

Ecdb (Oct. 2023). *European eCommerce Market 2023: Growth, Size, Revenue*. [Online; accessed 12. Oct. 2023]. URL: https://ecommercedb.com/insights/european-ecommerce-market-worth-us-1-1-trillion-by-2026/3982.

Edwards, L. & Veale, M. (2017). Slave to the algorithm? Why a'right to an explanation'is probably not the remedy you are looking for. *Duke L. & Tech. Rev.* 16, p. 18. DOI: https://dx.doi.org/10.2139/ssrn.2972855.

EuropeanParliament (2022). *The impact of the General Data Protection Regulation (GDPR) on artificial intelligence | Panel for the Future of Science and Technology (STOA) | European Parliament*. [Online; accessed 28. Oct. 2023]. URL: https://www.europarl.europa.eu/stoa/en/document/EPRS_STU(2020)641530.

EuropeanParliament (n.d.). *Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases*. [Online; accessed 13. Nov. 2023]. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A31996L0009.

FINLEX (2023). *Tekijänoikeuslaki*. [Online; accessed 28. Oct. 2023]. URL: https://www.finlex.fi/fi/laki/ajantasa/1961/19610404.

FINLEX (n.d.). *Copyright Act*. https://www.finlex.fi/en/laki/kaannokset/1961/en19610404.pdf. [Accessed on 11/13/2023].

Google, I. (2023). *Puppeteer | Puppeteer*. [Online; accessed 12. Oct. 2023]. URL: https://pptr.dev.

Goyal, K. (2023). *mechanize — mechanize 0.4.8 documentation*. [Online; accessed 13. Oct. 2023]. URL: https://mechanize.readthedocs.io/en/latest/index.html.

Heydon, A. & Najork, M. (1999). Mercator: A scalable, extensible Web crawler. *World Wide Web* 2(4), pp. 219–229. ISSN: 1573-1413. DOI: 10.1023/A:1019213109274.

Horch, A., Kett, H. & Weisbecker, A. (2015). Mining E-commerce Data from E-shop Websites. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 2, pp. 153–160. DOI: 10.1109/Trustcom.2015.575.

Intersoft (2022). *General Data Protection Regulation (GDPR)*. [Online; accessed 13. Oct. 2023]. URL: https://gdpr-info.eu.

Ismail, M., Ibrahim, M. M., Sanusi, Z. M. & Nat, M. (2015). Data Mining in Electronic Commerce: Benefits and Challenges. *International Journal of Communications, Network and System Sciences* 08(12). DOI: 10.4236/ijcns.2015.812045.

Jordan, M. I. & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science* 349(6245), pp. 255–260. ISSN: 0036-8075. DOI: 10.1126/science.aaa8415.

Jorge, O., Pons, A., Rius, J., Vintró, C., Mateo, J. & Vilaplana, J. (2020). Increasing online shop revenues with web scraping: a case study for the wine sector. *Br. Food J.* 122(11), pp. 3383–3401. DOI: 10.1108/BFJ-07-2019-0522.

Kim, C. (2018). *Web Scrape with Node.js and Cheerio - Christopher Kim - Medium*. URL: https://medium.com/@chriskimdevelop/web-scrape-with-node-js-and-cheerio-42a3123744f1.

Kosala, R. & Blockeel, H. (2000). Web Mining Research: A Survey. *SIGKDD Explor. Newsl.* 2(1), pp. 1–15. ISSN: 1931-0145. DOI: 10.1145/360402.360406.

Kouzis-Loukas, D. (2016). *Learning Scrapy: learn the art of efficient web scraping and crawling with Python*. 1st ed. Birmingham: PACKT Publishing. ISBN: 1784399787.

Kozinets, R. V. (2002). The Field behind the Screen: Using Netnography for Marketing Research in Online Communities. *Journal of Marketing Research* 39(1), pp. 61–72. ISSN: 0022-2437. DOI: 10.1509/jmkr.39.1.61.18935.

Krotov, V., Johnson, L. & Silva, L. (2020). Tutorial: Legality and Ethics of Web Scraping. *AIS Electronic Library (AISeL)* 47(1), p. 22. DOI: 10.17705/1CAIS.04724.

Krotov, V. & Silva, L. (2018). Legality and Ethics of Web Scraping. In.

Lekh, N. (2023). *Playwright vs. Selenium: which wins at web scraping?* URL: https://blog.apify.com/playwright-vs-selenium-webscraping.

lxml - Processing XML and HTML with Python (Dec. 14, 2022). [Online; accessed 13. Oct. 2023]. URL: https://lxml.de.

Ministry-of-Interior-Finland (n.d.). *Cybercrime*. [Online; accessed 28. Oct. 2023]. URL: https://intermin.fi/en/police/cybercrime.

Mitchell, R. (2018). *Web Scraping with Python: Collecting More Data from the Modern Web*. Sebastopol, CA, USA: O'Reilly Media. ISBN: 978-1-49198557-1.

Mithas, S., Ramasubbu, N. & Sambamurthy, V. (2011). How Information Management Capability Influences Firm Performance. *MIS Quarterly* 35(1), pp. 237–256. ISSN: 02767783. URL: http://www.jstor.org/stable/23043496.

Munzert, S., Rubba, C., Meißner, P. & Nyhuis, D. (2014). *Automated Data Collection with R*. ISBN: 978-1-11883481-7. DOI: 10.1002/9781118834732.

Olston, C. & Najork, M. (2010). *Web Crawling*. New York, NY, USA: Now Publishers. ISBN: 978-1-60198322-0.

Peelen, E. & Beltman, R. (2013). *Customer relationship management*. 2nd ed. Harlow: Pearson Education. ISBN: 978-0-273-77495-2.

Richardson, L. (2023). *Beautiful Soup: We called him Tortoise because he taught us.* [Online; accessed 12. Oct. 2023]. URL: https://www.crummy.com/software/BeautifulSoup.

Ridzuan, F. & Wan Zainon, W. M. N. (Jan. 1, 2019). A Review on Data Cleansing Methods for Big Data. *Procedia Comput. Sci.* 161, pp. 731–738. ISSN: 1877-0509. DOI: 10.1016/j.procs.2019.11.177.

Rust, R. T. & Huang, M.-H. (2014). The Service Revolution and the Transformation of Marketing Science. *Marketing Science* 33(2), pp. 206–221. DOI: 10.1287/mksc.2013.0836. URL: https://doi.org/10.1287/mksc.2013.0836.

Schwartz, B. (2004). *The Paradox of Choice: Why More Is Less*. HarperCollins Publishers. ISBN: 0060005696.

Scrapy (2023). *Scrapy Architecture overview*. [Online; accessed 15. Nov. 2023]. URL: https://docs.scrapy.org/en/latest/topics/architecture.html.

Selenium (2023). *The Selenium Browser Automation Project*. [Online; accessed 12. Oct. 2023]. URL: https://www.selenium.dev/documentation.

Shaikh, A., Khan, R., Panokher, K., Ranjan, M. K. & Sonaje, V. (2023). E-commerce Price Comparison Website using Web Scraping. *IJIRMPS - International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences* 11(3). ISSN: 2349-7300. DOI: 10.37082/IJIRMPS.v11.i3.230223.

Taleb, I., Serhani, M. A., Bouhaddioui, C. & Dssouli, R. (2021). Big data quality framework: a holistic approach to continuous quality management. *J. Big Data* 8(1), pp. 1–41. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00468-0.

Turban, E., Outland, J., King, D., Lee, J. K., Liang, T.-P. & Turban, D. C. (n.d.). *Electronic Commerce 2018*. Cham, Switzerland: Springer International Publishing. ISBN: 978-3-319-58715-8. URL: https://link.springer.com/book/10.1007/978-3-319-58715-8.

Turk, K., Pastrana, S. & Collier, B. (2020). A tight scrape: methodological approaches to cybercrime research data collection in adversarial environments. In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, pp. 07–11. DOI: 10.1109/EuroSPW51379.2020.00064.

Voigt, P. & Bussche, A. von dem (2017). *The EU General Data Protection Regulation (GDPR)*. Cham, Switzerland: Springer International Publishing. ISBN: 978-3-319-57959-7. URL: https://link.springer.com/book/10.1007/978-3-319-57959-7.