MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Webscraping tool for automating checkout process

Bachelor's Thesis

VLADIMÍR VITTEK

Brno, Fall 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Webscraping tool for automating checkout process

Bachelor's Thesis

VLADIMÍR VITTEK

Advisor: RNDr. Karolína Dočkalová Burská

Department of Computer Systems and Communications

Brno, Fall 2022

FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Vladimír Vittek

Advisor: RNDr. Karolína Dočkalová Burská

Acknowledgements

I would like to thank my supervisor, RNDr. Karolína Dočkalová Burská, for her continued guidance and patience. Additionally, I would like to thank my family and friends for their support during the whole course of my study.

Abstract

This bachelor's thesis aims to create an application for automating the checkout process on selected websites by utilizing web scraping techniques. The first part of the thesis is focused on exploring the background of web scraping, followed by explaining the motivation for web scalping. The legality analysis and the summary of the ethics of web scraping are provided to the reader. An analysis of existing solutions will be carried out to help aid the design and main requirements for a new desktop application. The second part focuses on designing and implementing the application. A guidebook for the resulting application is provided.

Keywords

web scraping, web scalping, legal analysis, scalper bots, scalping, Python, requests

Contents

Introduction **1**

1 Web Scraping **2**
1.1 Definition . . . 2
1.2 Use of Web Scraping . . . 2
1.3 Drawbacks of Using an API . . . 3
1.4 Process of Scraping Data . . . 3
1.4.1 Acquiring the Web Resources . . . 3
1.4.2 Parsing the Data . . . 4

2 Web Scalping **5**
2.1 Scalping . . . 5
2.2 Methods of Scalping . . . 6
2.2.1 Manual Scalping . . . 6
2.2.2 Scalper Bots . . . 6
2.3 The Effects of Web Scalping . . . 6
2.3.1 Bot Protection . . . 6
2.4 Existing solutions . . . 7

3 Legality of Web Scraping **9**
3.1 Legal Liability . . . 9
3.2 Legislation . . . 9
3.2.1 United States . . . 9
3.2.2 European Union . . . 12

4 Ethics of Web Scraping **14**
4.1 Moral Principles . . . 14
4.2 Conclusion . . . 16

5 Utilizing HTTP **18**
5.1 Request . . . 18
5.1.1 Request Line . . . 18
5.1.2 Headers . . . 18
5.1.3 Cookies . . . 19
5.1.4 Proxies . . . 19
5.2 Response . . . 20

5.2.1 Status Line 20
5.3 Capturing Network 20

6 Design **22**
6.1 Requirements 22
6.1.1 Functional Requirements 22
6.1.2 Non-Functional Requirements 23
6.2 Graphical User Interface Prototype 24
6.3 Websites Selection 26
6.4 Used Technologies 26
6.4.1 User Interface 26
6.4.2 Python HTTP Libraries 27
6.4.3 SQL Libraries 28

7 Implementation **30**
7.1 Frontend 30
7.1.1 Introduction to the User Interface 30
7.1.2 Validation 32
7.2 Backend 33
7.3 Module 34
7.3.1 Websites Support 34
7.3.2 Modeling Behaviour 34
7.3.3 Using a Captcha-Solving Service 36
7.3.4 Other Anti-Scalping Measures 36
7.4 Further Improvements 37

8 Conclusion **40**

A Application Manual **41**

Bibliography **42**

List of Figures

6.1 Tasks section design 24
6.2 Profiles section design 25
6.3 Proxies section design 25
7.1 Validation of user input in Create Profile dialog window . 32
7.2 Entity Relationship Diagram 33
7.3 Activity Diagram 35
7.4 Anti-scalping measure at Buzzsneakers 37

Introduction

Nowadays, any online store selling limited products can be confronted with a scalper bot. A scalper bot is a program designed to automate the process of purchasing goods. Scalper bots utilize web scraping techniques to achieve the fastest performance. They are able to complete the checkout process significantly faster than any human being. Their purpose is to buy as many of the limited items as possible, which is achieved by running in multiple threads using different billing and shipping information. Purchased products are then resold on secondary marketplaces at higher prices. Many websites deploy bot protections to fight bot activity.

Despite that scalping and scalper bots are growing phenomena, there is no literature focused on the legal background and design of these programs. The purpose of this thesis is to clarify legal boundaries of web scraping and create an application capable of purchasing limited goods on selected websites.

In the first chapter, there is a definition of web scraping along with a description of the process of collecting data from websites.

In the second chapter, the term web scalping is explained, followed by methods of scalping and its effects. The chapter ends with an analysis of the existing solutions of scalper bots, that points out their shortcomings.

In the following two chapters, legal and ethical aspects of web scraping take place. These chapters include studying the laws and court cases that touch upon web scraping. In the end, there is a summary of the moral principles of web scraping.

The fifth chapter focuses on Hypertext Transfer Protocol and its features needed to build a scalper bot.

The sixth chapter defines the functional and non-functional requirements of the application and introduces the mock-up of the graphical user interface. This chapter also describes the process of selecting technologies and comparison of the most relevant options.

The seventh chapter focuses on the implementation part of the thesis. It describes the difficulties encountered during the development. The chapter ends with the concepts of possible extensions.

Lastly, the eighth chapter concludes this thesis.

1 Web Scraping

1.1 Definition

Web scraping is a process of extracting data from the World Wide Web in an automated way. Data can be gathered manually via a web browser or by the HTTP protocol. Retrieved data is saved to a file or database for further processing and analysis. In practice, there are many ways to interpret this process, such as web crawling, data scraping, screen scraping, and many others. General consensus favors using web scraping [1], so that is the term that I am going to use in my thesis.

1.2 Use of Web Scraping

Web scraping is a powerful technique for assembling unstructured data from websites. It is a form of data mining and its goal is to transform collected data into structured data sets like spreadsheets, comma-separated value files, or a database.

Data like product pricing, product availability, stock pricing, and different reports can be collected through web scraping. Scraped data from websites can help you take constructive decisions in your business [2].

In practice there is a fair amount of scrapers in different fields, whose purpose can be:

- Scraping a product page of a retailer to spot if the product got discounted
- Scraping a product page of an unavailable product to spot if the product restocked
- Compare prices of the same product on different websites
- Monitor weather data
- Collect properties listings from real estate websites
- Collect government data
- Market analysis
- Monitor website changes

1.3 Drawbacks of Using an API

Web scraping would not be necessary if every site provided its own API. It is always better to use an API rather than coding a bot that collects the identical data, that is offered by the API. Not only that APIs are faster, but they also return pure data in JSON or XML format, which results in less work with parsing and formatting collected data. However, there are sites, which do not provide API or their API is not sufficient enough. This could be caused by various reasons:

- The data is protected or was not intended to be spread widely
- The data you want to scrape was not important to the creator and he did not include it in his API
- The web page does not have the technical ability or need to create an API

Another disadvantage to using an API is that it can rate limit users, which can make your script significantly slower if you need to collect real-time data.

At this point, web scraping comes in handy. You can browse the site as if you visited it from your browser and gather all the data you need [3].

1.4 Process of Scraping Data

The scraping process can be divided into two individual steps: acquiring the web resources and parsing the data.

1.4.1 Acquiring the Web Resources

There are two possible options for requesting a web page. The first one is building an *HTTP request* using requests library, urllib3, or other modules which let you send HTTP requests. HTTP request libraries offer to set your own cookies, authentication, and payload. Every scraping script starts with an HTTP request, which is sent to a web server for a specific page. Its purpose is to fetch the content of that specific page. Once the request is processed the web server will send back its response. The response can be in multiple formats:

- HTML and CSS
- XML or JSON data sets

- multimedia data (images, videos, sounds)

The second option is using *Selenium*, which is an alternative web browser controlled by commands. It looks like a common browser and provides automating the navigation on the site by programming which button should be clicked or what text should be typed into text areas. However, it fails if the elements are not loaded in time. Another disadvantage is that sites can easily notice any Selenium scraper on their web page and block them automatically. Next thing is that running multiple threads simultaneously causes a heavy load on your local machine or virtual server where you run your scraper. This means that you can run very few threads at the same time and monitor a smaller amount of data. Selenium could be used for a simple script that runs in a very small amount of threads and does not require a complex approach to requested data [1].

1.4.2 Parsing the Data

After the HTML file is retrieved, data can be extracted using either regular expression matching alone or in combination with additional logic. An alternative to this solution is using HTML parsing libraries, which work over the Document Object Model structure of the web pages. The most common Pythonic parsing libraries are BeatifulSoup and PyQuery. They provide functions for navigating, searching, and modifying parsed HTML trees.

The best practice is to keep matching expressions and paths to elements as general as possible to avoid possible vulnerabilities to changes in HTML files [4].

2 Web Scalping

Web scalping is a variation of web scraping and it refers to a practice that is used in reselling business. Even though web scalping has been known for several years, there is a small amount of literature, that would cover the whole concept. Most of the articles on the Internet are incomplete or even misleading. In this chapter, I am going to use my 5 years of experience in this field to explain the purpose, methods, and effects of web scalping.

2.1 Scalping

Scalping is a technique of making money by reselling something in limited supply. A scalper buys items at retail price and then resells them for a profit on a secondary market(eBay, consignment stores, local selling platforms). Scalpers focus on items with low stock and high demand [5].

Scalping started with reselling tickets to special events, such as festivals, concerts, or sports games. Scalpers bought as many tickets as they could and then tried to sell them last minute at the event venues for a higher price [5].

Tickets are not the only commodity that is targeted by scalpers. Recently, scalping emerged in various businesses. The ongoing global chip crisis, which started back in 2020 during COVID-19 and is present to this day [6], caused shortages in stock for graphic cards and gaming consoles, which caused their value to rise on secondary markets. Some even resell luxurious watches, collectible cards, or limited editions of alcohol bottles. However, the most common thing to resell has always been shoes and clothes. Corporates, such as Nike or Adidas, decided to upscale their business by releasing fashionable silhouettes of sneakers in limited stock numbers, which would help them promote their brand and increase sales of other models, that are produced in large numbers.

2.2 Methods of Scalping

2.2.1 Manual Scalping

The most obvious way of acquiring these items is to act as a regular customer and try to purchase the limited item manually. This option is not very successful, because of competition from other customers and bots [5].

2.2.2 Scalper Bots

A scalper bot is a program designed to automate the checkout process on a selected website. The user sets up the scalper bot for the anticipated release of a limited product and aims to place as many orders as he can. Scalper bots run their purchase script in multiple threads, which are called tasks, to achieve multiple successful checkouts. These bots scrape the product page in short intervals and check if the product is in stock. When the sale starts and the product becomes available, they proceed to cart the product and place the order. Compared to a manual user, a scalper bot can finish checkout significantly faster [7].

2.3 The Effects of Web Scalping

Scalping makes getting items in limited supply harder for regular customers. Scalpers buy these items in large quantities and sell them to consumers at a higher price. This can lead to retailers losing their loyal customers or hurting their reputation, because customers experienced badly organized online releases, that allowed scalper bots to acquire multiple items. For that reason, retailers implement anti-scalping measures on their websites. This happens to be mainly limiting one product per customer and implementing bot protection [7].

2.3.1 Bot Protection

There are several effective serves-side bot management software tools for preventing scalper bots from accessing the website. In practice, there are three basic strategies: *activity tracking*, *blacklists*, and *challenges*. Websites implement all of them to achieve the best possible protec-

tion. The best-known bot protection services are Captcha, Cloudflare, Akamai, Datadome and PerimeterX [8].

Activity Tracking

Its purpose is to identify traffic that originated from a scalper bot and block its access. This strategy uses an AI system that rates requests based on their credibility. Occasionally, this strategy can raise some false positives and block genuine visitors [8].

Blacklists

Another strategy is to filter out IP addresses with suspicious activity. Scalpers use multiple IP addresses to run their bot in multiple threads, while each thread uses a unique IP address. Scalpers usually share subnets of their IP addresses, because they purchase them from the same providers. These known bot IP subnets can be found in *blacklists*, which can be either compiled by non-profit organizations or large cybersecurity companies that offer spam filters and DDoS protection services [8].

Challenges

Challenges are useful in blocking automated submissions by scalper bots. Even though bots are able to solve this challenge using a third-party software, it ensures that spamming the website costs the scalper resources. On the other hand, the general public gets annoyed by the "I am not a robot" challenges and the need to distinguish themselves from automated software [8].

2.4 Existing solutions

The opportunity for huge profits to be made from scalping created a new industry focused on developing scalper bots. Scalper bots licenses are sold in limited quantities and they are being resold as well. The average bot's retail price ranges from $300–$500 and if a scalper cannot get it for retail, he has to pay resale value and it can cost up to $1000–$8000 excluding the monthly subscription fees [9].

The development is often sped up by the need for the fast delivery of the final product, which is one of the reasons for the downsides of these programs. In this section, I am going to point out these inconveniences and scheme out missing features.

User Interface

The majority of scalper bots use a command line interface to maximize their performance. However, it makes the user's experience intricate. Not only that the user has to set up the bot in a text editor, but also the status of individual tasks is hard to track since the running application uses a single terminal window to print all statements.

Limited Control

The command line architecture leads to limiting the user control over the running scalper bot. The only way the user can control the process is to stop the bot by closing the terminal window. The user is not able to start or stop selected tasks. Moreover, there is no ability to edit the delay of tasks while the bot is running, which is in my opinion a feature that would be useful for scalpers.

Lack of Website Support

Each scalper bot has a list of websites, which are supported. Developers of these bots focus on well-known retailer shops, that receive a bigger quantity of limited products, to catch the interest of more scalpers, that would purchase their bot. This marketing behavior leaves some of the smaller retailers' websites with zero bot competition.

3 Legality of Web Scraping

The legal aspect of web scraping is in a grey area. Therefore, a user must be aware of possible unlawful behavior when he collects data and also the way he works with it. Although this topic would be suitable for a whole thesis, I am going to summarize the basic principles and court cases in this chapter to give a user an overview of laws on different continents.

3.1 Legal Liability

When a user commits an illegal act while using a web scraping tool, he is fully responsible for his actions. Although the creator provided the user with this tool, he is not responsible for its use by other users. The creator can be held responsible when the tool was designed explicitly for illegal activity. It is recommended to enlighten the user about the legal aspect of web scraping before giving him access to a scraper [10].

3.2 Legislation

To this date, there is no legislation that would address web scraping directly. However, web scraping could possibly break several laws, which were accepted before web scraping rose in popularity. Those who scrape websites on the Internet balance in an uncertain legal landscape. Despite that, several court cases around the world have helped to clarify what is acceptable and what is not.

3.2.1 United States

In the United States, web scraping legality is judged based on fundamental cybersecurity laws and legal theories such as the CFAA[1], copyright infringement, breach of contract, and trespass to chattels [11].

1. Computer Fraud and Abuse Act

Computer Fraud and Abuse Act

Obtaining data through web scraping and using it for fraudulent or illegal purposes is prohibited by several laws. The CFAA is the most common codex used in web scraping cases. It forbids individuals from intentionally accessing a protected computer without authorization or exceeding authorized access [11].

This statute is applicable to hiQ v. LinkedIn data scraping case. In 2017, HiQ, a data analysis company, filed a lawsuit against LinkedIn for restricting hiQ's access to LinkedIn's website. HiQ, which collected data from LinkedIn user profiles, argued that this behavior is anti-competitive and violates federal laws. On the other side, LinkedIn stated that hiQ was not entitled to an injunction, since their behavior is in conflict with CFAA as a result of hiQ's unauthorized access to LinkedIn's website. The court clarified that accessing a computer without authorization is violated only when a person gets around a computer's access permission, such as username and password. They also stated that when a website owner permits public access to a website's data, a user accessing that data will not be in conflict with CFAA from a legal standpoint [12].

In contrast, people who were prosecuted under CFAA had knowingly accessed premium content and then resold it or continued accessing it via an unauthorized channel [11].

Breach of Contract

Website owners often argue that they can prohibit the scraping of their website by explicitly forbidding the use of scalpers in the *Terms of Use* policy. The user has to agree with the website's policy when he first enters the website. Scrapers encounter brief summary of scraping rules via *robots.txt* file (explained in Chapter 4). Although breaking these terms may lead to a breach of contract on the user's side, it cannot lead to legal accusations against the scraper. Furthermore, the website would have to prove that the scraper caused material damage to the site [11].

Trespass to Chattels

If a website is damaged or overloaded by a scraper, then the person, who runs the scraper can be held liable under the trespass to chattels tort [11].

In 2000, the online auction site eBay sued Bidder's Edge, a site that gathered data from various auction sites to aggregate all auctions into one place. Bidder's Edge obtained this data using web scraping and provided this information to its users. Bidder's Edge was sued under multiple theories and one of them was trespass to personal property. eBay claimed that Bidder's Edge committed a trespass when it used, accessed, and intermeddled with eBay's systems without eBay's permission and knowledge. Bidder's Edge's defense was built on the fact that the site is available for everyone, but their argument was unconvincing. Despite Bidder's Edge's minimal use of eBay's system capacity, the court found the injury sufficient enough to state a claim for relief [13].

After this exceptional lawsuit, courts have ruled that the damage needs to be material and easy to prove in order for the website owner to be eligible for financial compensation. Due to the fact, that damage is usually hard to prove, the trespass to chattels is rarely used in web scraping lawsuits [11].

Copyright Infringement

Data that is owned and copyrighted by the website owner cannot be scraped or republished. Doing so can lead to copyright infringement, especially when the data is collected for financial gain. On the other hand, copyright laws do not include user-generated data (e.g. content from social media). For instance, a website that allows users to review products does not necessarily own the reviews written by its users. Additionally, scrapers can use copyrighted data to create summaries of this data.

Scrapers can also collect copyrighted data on a limited scale under the "fair use" principle. This principle is still a grey area. On one hand, several courts decided that scrapers made fair use of scraped material, but on the other hand the court in the Meltwater case came to a conclusion that even the use of a very small amount of copyrighted data

(4,5%) could be enough to exceed the "fair use" border. Nevertheless, it is important to note, that the data in this case was meant to be purchased. If the data was available to the public, the judge might have come to a different conclusion [11].

Trade Secrets

Web scraping should not be used to develop a monitoring mechanism to reveal trade secrets in competing companies [11]. A trade secret could be a piece of information that has either actual or potential independent economic value or has value to others who cannot legitimately obtain the information [14].

In 2017, Waymo, a self-driving car company, filed a lawsuit against Uber for stealing trade secrets using web scraping. Uber scraped pricing information and gathered vehicle location data from competitors' applications. Uber then used machine learning to analyze how many trips competitors were making, which would reveal the market share dominated by competitors [15].

3.2.2 European Union

Web scraping's legal perspective in the European Union is more focused on protecting the website's data than protecting the website itself. There have not been a lot of court cases that touched upon web scraping, but it is important to analyze them, since in the European Union, courts can decide based on the result of previous similar cases.

General Data Protection Regulation

The GDPR is a set of regulations that protect the personal information of citizens in the European Union. It ensures that EU citizens are treated fairly when it comes to collecting their personal data. GDPR defines several regulations defining how companies process the data, store the data, and the consequences of breaches of personal data. The GDPR and web scraping cross paths when you collect personal data of EU residents, such as names, pictures, IP addresses, e-mail addresses, etc. Regarding the legality of processing personal data, a scraper must comply with one or more of the following legal reasons [16]:

- Data subject has given consent to the processing of his or her personal data.
- Processing is necessary for the performance of a contract to which the data subject is a party.
- Processing is necessary for compliance with a legal obligation.
- Data processing is in the scraper's legitimate interest or for the public good.

Database Rights

In the European Union, collecting data from a database is regulated by Directive 96/9/EC. This legal framework provides copyright or *sui generis*[2] right protection to qualifying databases. It also defines mandatory rights for lawful users.

A frequently cited example is that of Ryanair v. PR Aviation. PR Aviation is a Dutch website providing a comparison of flight prices. This information was provided by scraping flight data from other companies, including Ryanair. However, Ryanair's Terms of Use strictly prohibited the use of automated software to extract data for commercial purposes, unless the third party has concluded an agreement with Ryanair. Ryanair claimed that PR Aviation violated its database rights under Directive 96/9/EC and breached its Terms of Use. The Netherlands Supreme Court dismissed Ryanair's claim for infringement of its database rights. However, it requested a preliminary ruling from The Court of Justice of the European Union (CJEU) on Ryanair's claim for breach of contract. The CJEU agreed that Directive 96/9/EC does not apply to databases that are not protected by copyright or the *sui generis* right. Consequently, it held that Directive 96/9/EC does not preclude the author of a database to set limitations on its use by third parties [18].

2. Sui generis means "of one's or its own kind". In law, this is a term of art to identify a legal classification that exists independently of other categorizations because of its uniqueness [17].

4 Ethics of Web Scraping

As explained in the previous chapter, there are some laws that guide you toward unharmful web scraping, but do not cover the whole specter of web scraping. There are basic principles, which should be respected and make the scraping process more transparent and ethical.

4.1 Moral Principles

Preferring an API

Using an API is always the best choice when it provides the data you want to collect. Not only it is easier, but you can be sure that you are scraping the data in a way that website approves and acknowledges [19].

Terms of Use

Read the *Terms of Use* of the website and act accordingly to it. The website has the right to block your access if you do not respect its policy. Even though there is no possibility of a lawsuit, respecting the rules of a website is the right way to collect data in an ethical way [19].

Robots Protocol

Follow the instructions in *robots.txt* file. This file can be found in the website's root directory and is considered a Terms of Use for robots. The *robots.txt* file starts with one or more User-Agent fields, that specify which robots the rules apply to, followed by the rule fields, such as Allow or Disallow, showing the robot which parts of sites are allowed or disallowed to scrape.

Here is an example of Youtube's robots.txt file extracted from `https://youtube.com/robots.txt`:

```
# robots.txt file for YouTube
# Created in the distant future (the year 2000) after
# the robotic uprising of the mid 90's which wiped out all humans.
```

```
User-agent: Mediapartners-Google*
Disallow:

User-agent: *
Disallow: /comment
Disallow: /get_video
Disallow: /get_video_info
Disallow: /get_midroll_info
Disallow: /live_chat
Disallow: /login
Disallow: /results
Disallow: /signup
Disallow: /t/terms
Disallow: /timedtext_video
Disallow: /verify_age
Disallow: /watch_ajax
Disallow: /watch_fragments_ajax
Disallow: /watch_popup
Disallow: /watch_queue_ajax

Sitemap: https://www.youtube.com/sitemaps/sitemap.xml
Sitemap: https://www.youtube.com/product/sitemap.xml
```

It shows that "Mediapartners-Google" scrapers have no restrictions, while other bots should not visit directories and files matching "/comment", "/get_video", "get_video_info", ...

It is important to note, that a user is not legally required to follow these instructions, but the ethical norms say that scrapers should follow them [11, 19, 20].

User-Agent

Add a User-Agent header to your scraper. It serves as a characteristic string that lets the website know the application, operating system, vendor, and version of the requesting User-Agent. Doing so will make the scraping process more transparent [19].

Refresh Rate

Use high delays between requests. Scraping the site using a reasonable refresh rate is a good practice. Some websites even define minimum delay between requests in their *robots.txt* file. In most cases, the user is asked to use at least 10 seconds delay between each request. If the refresh delay is too low, your scraper could cause damage to the website or be confused for a DDoS attack [19].

Data Selection

Collect only the data you need. Be cautious about the scraped data and be reasonable when selecting the information that you gather [19].

Data Storing

Respect and protect the scraped data. This involves storing it safely, not distributing it without authorization, or using it as your own. This rule serves not an only ethical purpose, but can also prevent any legal issues [19].

Communication

Respond to the website's objections and be ready to communicate. Website owners can change their minds and scraping rules might change [19].

4.2 Conclusion

Based on these regulations, a user developing a scraper should consider some questions regarding his/her data scraping actions:

- Is there an alternative in form of an API that satisfies your data requirements?
- Is web scraping prohibited by the website's *Terms of Use*?
- Is your scraper in accordance with *robots.txt* instructions?
- Is there a chance that your scraping activities could cause material damage or financial loss?

- Did the website contact you and asked to stop your scraper, blocked your IP address, or restricted your access to its data in some way?
- Is it possible that your scraping activities harm individuals, communities, or organizations?

A positive response to any of these questions may indicate that your project might be breaking moral principles [11].

5 Utilizing HTTP

In this chapter, I am going to introduce the most important segment of web scraping – Hypertext Transfer Protocol (HTTP), focusing on its features that are fundamental for building a web scraper.

The HTTP is a simple networking protocol. It is text-based, which makes it easily readable for end users. It follows a simple request-reply communication scheme. The main component in the exchange of messages between a client and a web server is an HTTP request message, that is sent to a web server, followed by an HTTP response [21].

5.1 Request

A request message contains the following[21]:

- A request line,
- Request headers, each one on their own line,
- An empty line,
- An optional message body, which can be split into multiple lines.

5.1.1 Request Line

The request line consists of 3 parameters divided by space. The first one is the HTTP method, that we want to execute. The most common methods are *GET*, which is used to retrieve data from a web server, and *POST*, which is used to send data to a web server. The HTTP method is followed by the URL we want to interact with. Lastly, there is the HTTP version our program understands [21].

5.1.2 Headers

Every HTTP request and response between a user and a web server includes one or multiple headers. They provide additional information and consist of a case-sensitive name followed by a colon and value. If the headers are configured properly, they can serve as effective and cheap bot protection [22].

Therefore, when coding a scraper, it is mandatory to include these security headers with corresponding values in requests. For instance, a POST request for sending a form could include a *token* header, which value can be scraped from the form's page using a GET request.

User-Agent

The *User-Agent* header is a special header used for identifying the user, who is sending the request. It includes the name and version of the user's browser [21].

Some web servers can deny access to scrapers using a default User-Agent to forbid their usage on their site. The best way to get around this protection is to give this header a value of an existing browser's User-Agent.

5.1.3 Cookies

Despite that cookies are HTTP headers themselves(*Set-Cookie*), they are treated differently due to their broad usage. Cookies are a major mechanism for authentication and authorization for web servers. The *Set-Cookie* header is one of a few headers that can be found multiple times in one request or response [22].

A person, who is coding a scraper, does not need to include cookie headers in his requests, thanks to the scraping libraries, which do that automatically. However, he needs to execute these requests using a session to store them.

5.1.4 Proxies

Request can be executed with the use of proxies. An HTTP proxy server acts as an intermediary for HTTP requests. A user sends an HTTP request, which is first sent to the proxy server, which forwards it to a web server. Proxy servers are used for anonymity reasons, to mask the user's IP address [21].

The main reason behind masking your IP address is to prevent blocking the user's IP address by the web server when he sends multiple requests in a short time.

5.2 Response

After the web server processes the request, it sends back a response. It looks very similar to the HTTP request and consists of [21]:

- A status line,
- response headers, again each one on their own line,
- an empty line,
- an optional message body, which can be split into multiple lines.

5.2.1 Status Line

The first line in response starts with the HTTP version and continues with a status code and a status message, indicating whether the user's request succeeded or not [21].

Status Codes

Status codes offer an easy way for clients to understand the result of their requests. There are multiple established HTTP status codes, each of them with its own meaning. To have a better overview of status codes meaning, we divide them into 5 different classes [23]:

- 100-199: Informational Status Codes
- 200-299: Success Status Codes
- 300-399: Redirection Status Codes
- 400-499: Client Error Status Codes
- 500-599: Server Error Status Codes

5.3 Capturing Network

After learning about requests, there is a need to figure out how to capture them, to be able to replicate them. There are applications built to analyze network connections and capture network packets. The most popular ones are Fiddler, Wireshark, and Charles Proxy. I am going to explain the concept of the one that I am going to use – Fiddler.

Concept of Fiddler

Fiddler works as a special proxy server, that runs on its user's machine. Other programs, which are running on the user's computer send their HTTP requests to Fiddler, which forwards the traffic to a web server. The response from the web server is processed by Fiddler and then returned to the user. Due to the fact, that Fiddler captures traffic from all processes that are running locally, it supports a wide selection of filters, which lets the user highlight only the traffic that he wants to focus on [24].

The usage scenario for building a scraper would be to set up Fiddler to record traffic from a web browser and manually execute the exact steps, which the user wants to replicate. After capturing these requests, the user can inspect them and extract the information he needs, which could be mainly headers, body, and the URL of requests, but also responses and their status codes.

6 Design

As mentioned in the analysis of current scalper bots (Chapter 2), the most significant deficiencies of the existing solutions are *chaotic user interface* and *missing features*.

I decided to develop a scalper bot that provides a user-friendly graphical interface in form of a desktop application. The main purpose of the decision to build a desktop application was to provide a straightforward overview of executed tasks and the ability to add more components to control the running process.

In this chapter, I will summarize functional and non-functional requirements followed by the user interface design and analysis of the checkout module. At the end of the chapter, I am going to introduce the technologies, that I will be using during implementation.

6.1 Requirements

6.1.1 Functional Requirements

F1. The application's graphical user interface shall be divided into a sidebar menu and a main window. The sidebar menu shall contain navigation items to specific sections.

F2. The application shall allow the user to display a list of tasks in the Tasks section.

F3. Starting each task shall start a new thread, that attempts to purchase a product from the product link defined by the user.

F4. The application shall allow the user to display a list of profiles in the Profiles section.

F5. The application shall allow the user to display a list of proxylists[1] in the Proxies section.

F6. The application shall provide an option to avoid using proxies.

1. a proxylist is referred to as a group of proxies.

F7. The application shall allow the user to display a list of checkouts in the Checkouts section.

F8. The application shall allow the user to edit settings in the Settings section.

F9. When the application is run for the first time, default settings shall be created.

F10. When the user edits the delay in the Settings section, this change shall apply to all tasks (even to those, that are running).

F11. The application shall be able to solve captcha challenges using a third-party service.

F12. The application shall validate user inputs.

6.1.2 Non-Functional Requirements

N1. The application shall use English.

N2. The application shall be coded in Python.

N3. The application shall provide a simple graphical user interface. Users with fundamental scalping knowledge should agree that the interface is clear, intuitive, and easy to use.

N4. The application shall store its data in a local SQL database.

N5. The application shall not require an installation.

N6. The application shall run on the client's desktop and does not require server-side support.

N7. The application shall be designed in a way that a developer can comfortably add new featured store modules.

N8. Stopping the task shall not take longer than the delay specified in the Settings section.

N9. The application shall not violate any laws and be legal in every aspect.

N10. The application shall follow the moral principles of web scraping.

6.2 Graphical User Interface Prototype

There is a need to build a desktop application with a graphical user interface to satisfy mentioned requirements. I decided to use Figma, an interface design tool, to create a mock-up of the main screens. I followed the requirements to build a clear graphical user interface including all the features to create, start, stop, edit or delete chosen elements. The following pictures present a prototype of the graphical user interface of the application. The Figure 6.1 shows the design of the Tasks section, the Figure 6.2 shows the design of the Profiles section and the Figure 6.3 shows the design of the Proxies section.

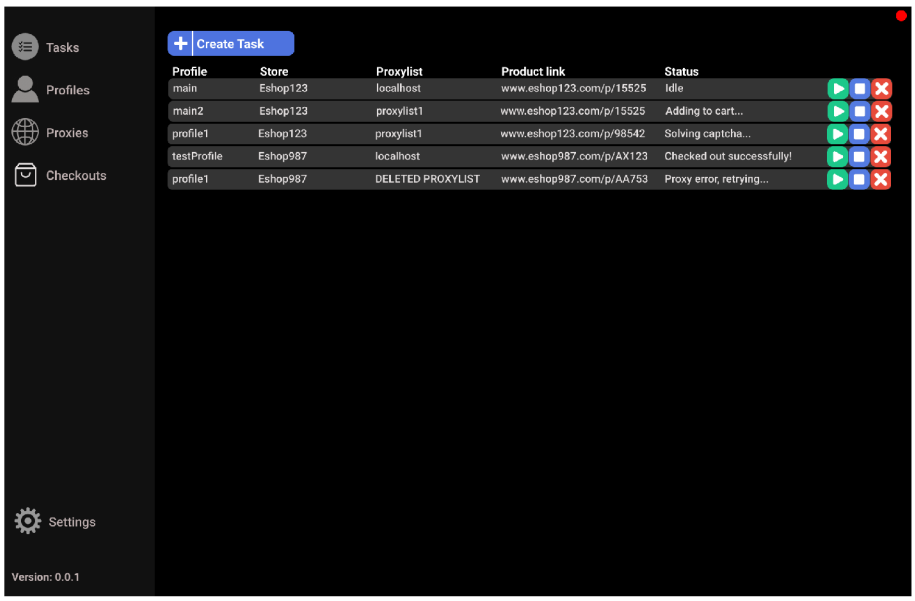

Figure 6.1: Tasks section design

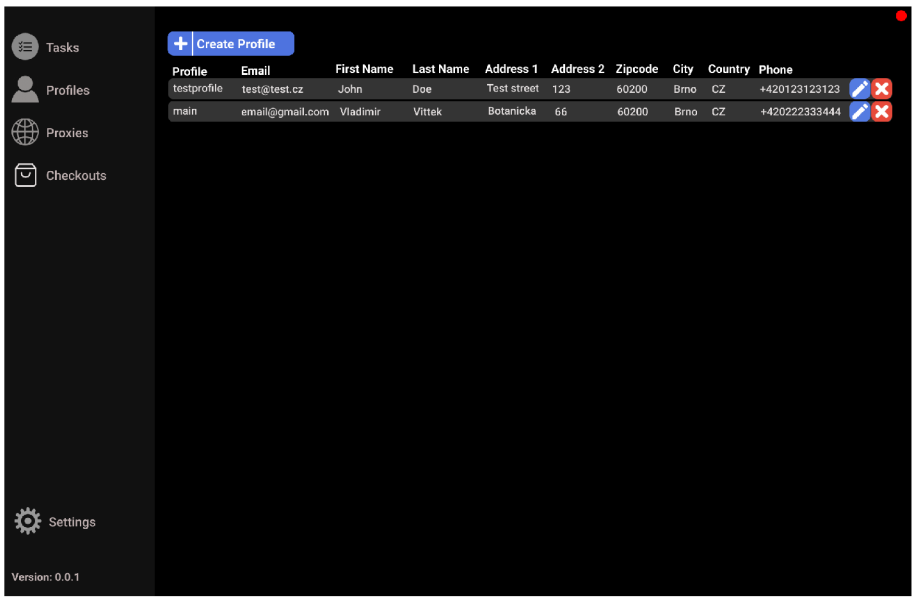

Figure 6.2: Profiles section design

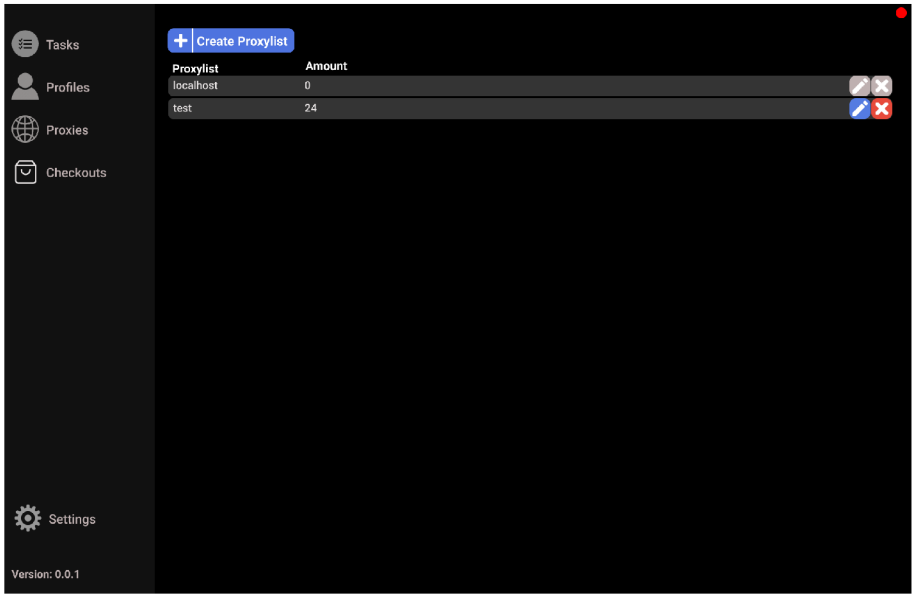

Figure 6.3: Proxies section design

6.3 Websites Selection

My goal is to select websites, that are not covered by existing scalping applications. I will focus on stores that are hosting limited releases of shoes in the Czech Republic and Slovakia. These retailers rarely receive the limited shoes and if they do, the stock numbers are really small. Nevertheless, it gives a greater chance of success due to no competition from other scalper bots.

Another specification is that the website needs to provide "cash on delivery" payment method to avoid dealing with credit card payment processors.

6.4 Used Technologies

6.4.1 User Interface

PyQt

PyQt is a toolkit for graphical user interface widgets. It uses Styles to achieve large customization and it is possible to change Styles while the application is running. It also supports the creation of shaped widgets.

The disadvantage to PyQt is that it was released under General Public License, which forbids a person from distributing software for commercial uses without including the source code [25].

TKinter

TKinter is the standard GUI framework that comes built into Python. The positive aspects are that it was released under a free software license and has a very simple syntax.

On the contrary, it allows only small customization of its widgets and it is not available to change styles during runtime. Another drawback is that shaped widgets are not supported [25].

Conclusion

It is clear that *PyQt* provides more features for designing. My application is not meant to be used for commercial use, therefore I am going to work with *PyQt*.

6.4.2 Python HTTP Libraries

Aiohttp

Aiohttp is an asynchronous HTTP Client/Server for Python's module asyncio. It supports the asynchronous programming concept, which can be helpful for running the script in multiple threads. It also features sessions and proxy support [26].

Urllib3

Urllib3 client brings more features to HTTP requests. The most significant ones are being able to reuse connections thanks to caching and the possibility of retrying requests. The use of proxies is supported as well.

On the other hand, urllib3 does not have session objects and instead of that forces the developer to manually include cookies in the headers [26].

Requests

Requests library is an elegant and straightforward HTTP library, that is highly favored within the Python community. The bright side of this library is that encoding and decoding of the data are done automatically. There is also no need to specify the Content-Type header compared to urllib3. Requests library also features both session and proxy support [26].

Conclusion

The main features I am looking for are definitely proxy and session support. Both *Requests* and *Aiohttp* satisfy this requirement. They differ in the approach to synchronous and asynchronous programming.

Table 6.1: Comparison of Python scraping libraries

Library	Proxy support	Session support	Async
Aiohttp	Yes	Yes	Yes
Urllib3	Yes	No	No
Requests	Yes	Yes	No

Despite that it may seem that asynchronous programming might be more appropriate for my solution, I am going to use *Requests* since the PyQt library does not support asyncio by default and needs a more complex approach to integrating asynchronous functions.

6.4.3 SQL Libraries

MySQL

MySQL is one of the best-known Relational Database Management System connectors. It uses server-client architecture and it performs well in terms of security, scalability and replication.

On the other side, MySQL does not handle bulk INSERT operations well [27].

SQLite

SQLite is one of the most popular and easy-to-use relational database systems. Unlike other mentioned libraries, SQLite does not use server-client architecture and instead stores its data in a file on the client's hard drive. It is used for small and simple projects due to its fast connection.

The downside is that SQLite's operations are serialized, which makes it weak for applications with multiple users [27].

PostgreSQL

PostgreSQL is another Relational Database Management System connector, that uses server-client database architecture. It focuses on high extensibility and is often used for research and scientific project databases [27].

Conclusion

As mentioned in requirement N4, I shall work with a serverless database. *SQLite* provides this feature along with fast connection speeds, so *SQLite* is my database of choice.

7 Implementation

This chapter is focused on fulfilling requirements from the previous chapter. I also mention the difficulties that I encountered during the development phase. The chapter ends with suggestions for future development.

7.1 Frontend

7.1.1 Introduction to the User Interface

Following requirements from the previous chapter, I divided the desktop frame into the sidebar menu and the main window. The sidebar menu contains the Tasks section, the Profiles section, the Proxies section, and the Settings section. Screenshots of these sections are included in the guidebook, which I mention in the Application Manual.

Tasks Section

In the Tasks section, the user can create new tasks and control existing tasks. To give the user better control of tasks I added buttons to start, stop and delete all tasks. The user is able to create a new task by clicking on the Create Task button and filling out the form. The form includes a store, a profile, a proxylist, a product link, and a size range in the EU sizing chart. The size range defines the acceptable sizes, that can be purchased using a checkout module of the selected store.

Profiles Section

The Profiles section provides the user with the ability to create a new profile, but also allows him to edit and delete existing ones. To create a new profile the user has to interact with the Create Profile button to open the dialog window. He has to fill out the form, which includes the profile name and all the information needed for creating an order on any website, such as email, address, phone number, etc.

Checkouts Section

The Checkouts section serves as an overview of checkouts. The user can see successful purchases, that come from the Tasks section. Each one includes the product name, the product price, the store, the proxylist, and the profile used to secure this product.

This section could serve as an analysis. If the user runs multiple proxylists during the release, he can conclude which proxylist worked the best based on the amount of checkouts for individual proxylists.

Proxies Section

The Proxies section is very similar to the Profiles section. It allows the user to create a new proxylist by clicking on the Create Proxylist button, where the user inputs the proxylist name and proxies. The user is able to edit or delete existing proxylists, except the proxylist named "localhost", which is the only proxylist that does not use any proxies. If the user wants to run a task without proxy support, he should select this proxylist in the Create Task dialog menu.

Settings Section

In the Settings section, the user can set his own delay, which works as a timeout for all running tasks. If a running task does not finish one of its steps successfully, it waits the time defined in the Settings section before retrying the step again. The change of this value is applied to all tasks instantly, even to the running ones.

The next feature is setting up a User-Agent, that will be assigned to all tasks. As mentioned in Moral Principles, a scraper script should include a User-Agent header that lets the site know the name of the scraper. There is also an option to use random User-Agents.

These two features partially satisfy the requirement to keep the application within moral principles.

Lastly, there is the input for the 2Captcha API key, which is mandatory to run the Popname module, since it requires the support of a captcha solver. A detailed description of handling the captcha protection will be discussed later in this chapter.

7.1.2 Validation

During my work, I needed widgets with specific constraints on their input values. PyQt provides a QValidator class along with QRegExpValidator class, which compares the user's input against a regular expression. Most of the user's inputs on the frontend of my application are QLineEdit widgets. However, I encountered an issue when I needed to validate a QTextArea widget. The validation classes provided by PyQt are compatible only with QLineEdit widgets, which forced me to implement my own validation.

As shown in the Figure 7.1, the final implementation of the forms includes the validation by regular expression and logical validation, which includes rejecting duplicate or empty names for profiles and proxylists.

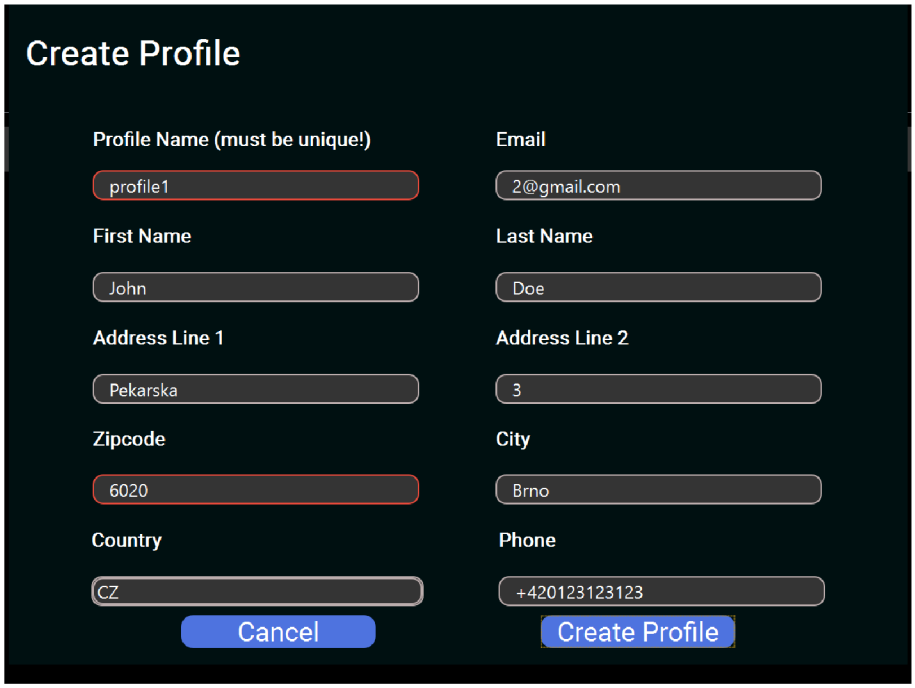

Figure 7.1: Validation of user input in Create Profile dialog window

7.2 Backend

The following figure of an Entity Relationship Diagram illustrates how the entities are related to each other.

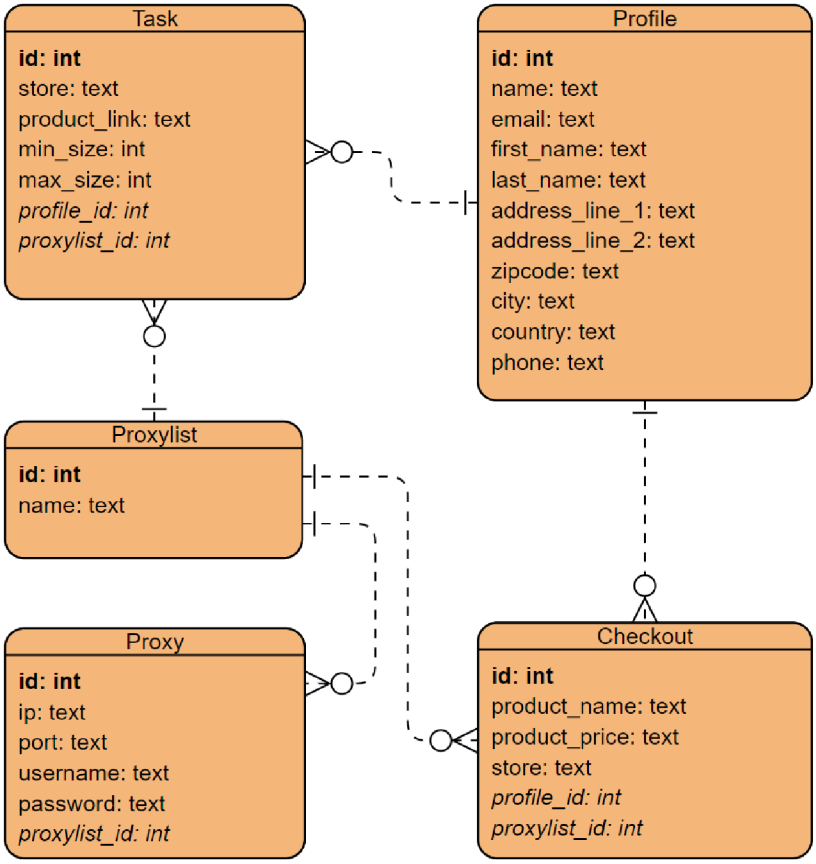

Figure 7.2: Entity Relationship Diagram

When the user starts the application for the first time, the database is created. Along with creating tables, the default settings and the proxylist "localhost" are inserted into the database. Each table has its own queries that provide functions to create, read, update and delete records from the database. Additionally, I worked with unittest library [28] to test the queries.

7.3 Module

7.3.1 Websites Support

The idea was to select the websites, which have not been targeted by publicly available scalper bots yet. They also needed to provide the "cash on delivery" payment method and provide shipping to both Slovakia and the Czech Republic. After examining the market of shoe retailers, I selected 3 of them that would match my requirements. I implemented the automated checkout process for these websites: Queens[1], Popname[2], and Buzzsneakers[3].

Queens and Buzzsneakers provide different domains for their customers from Slovakia and the Czech Republic, but they follow almost the same checkout steps. There are only small differences in requests' URLs, headers, and bodies. The task will select the module's variant by the profile's country. For example, running a task on Queens using a profile with a country set as Slovakia will require a product link from the Slovak domain.

Even though all the website modules are working at the moment of submission of my thesis, this could change over time. I cannot predict the changes implemented by the website owners, that could break my module. If a change like this occurs, the application will not crash and the status of the task running this module will be set to an error message.

Furthermore, I added a module called Testing, which serves the purpose to test the graphical user interface. The developer can create a task with the Testing store module to test features, such as starting, stopping, and deleting the task or changing the delay while the task is running.

7.3.2 Modeling Behaviour

The concept of the checkout module is to purchase the desired item from a website. The process, which is visualized in Figure 7.3, follows multiple steps to achieve a successful checkout:

---

1. `https://www.queens.cz/`, `https://www.queens.sk/`
2. `https://www.popname.cz/`
3. `https://www.buzzsneakers.cz/`, `https://www.buzzsneakers.sk/`

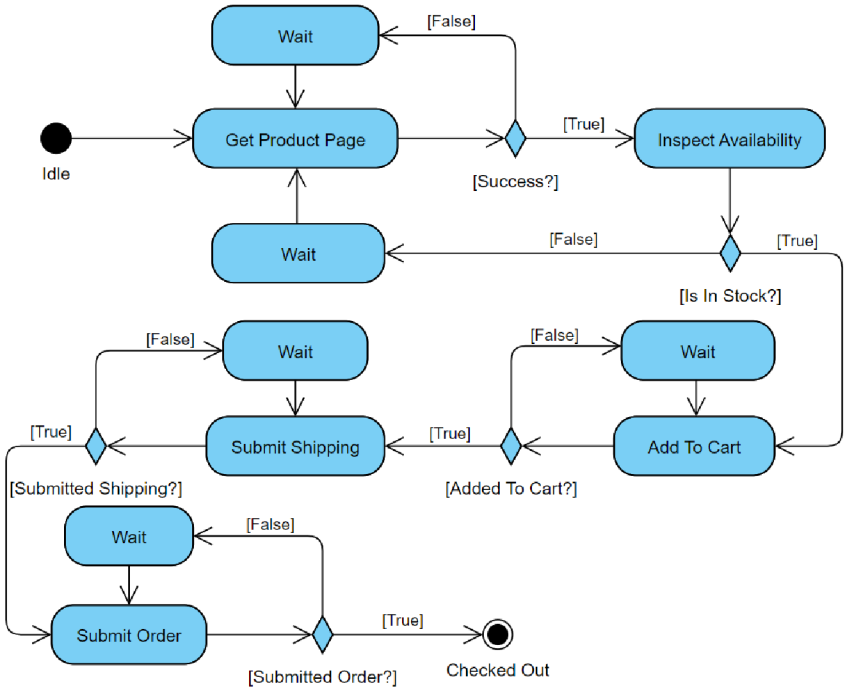

Figure 7.3: Activity Diagram

1. The module starts from "Idle" status and approaches to get a product page content and sets the task's status to "Getting product page".
2. After the product page content is scraped, the availability of the product is inspected. If the product is available, a post request to add the item to the cart is sent and the task's status is set to "Adding to cart". Otherwise, the task's status is set to "Out of stock" and the process from the first step is repeated.
3. After the item is successfully added to the cart, the module will proceed to submit shipping information defined in the task's profile and the task's status will be set to "Submitting shipping".
4. If the post request to submit shipping information is successful, the module will send a post request to place the order and the task's status will be changed to "Submitting order".

5. If the order is placed successfully, the status of the task will change to "Checked out". Detailed information about the purchase will be visible in the Checkouts section.

It is necessary to check if the request succeeds in every step. If it fails, the module will change the task's status to inform the user which step was not successful, wait for a period of time defined in settings and resend the request until it succeeds. Additionally, if the reason for the failed request was blocking the user's access to the website, then the proxy, that was being used, will be swapped for a different unused proxy, taken from the task's proxylist.

Regarding bot protection, if a post request includes a captcha bot protection, it should be handled. The task's status will be updated to "Solving captcha" when a captcha challenge is encountered. The handling shall be done using a third-party service as determined in requirement F11.

7.3.3 Using a Captcha-Solving Service

Among selected websites, there was one that used a captcha bot protection. Popname included a captcha submission at the end of the checkout process. For this reason, I needed to implement a captcha solver. A captcha-solving service is made to automate the process of solving captchas. The user sends a captcha challenge to the service's API, where it is distributed to a human worker. The worker solves this captcha and sends back the answer to the user. The user then uses this response in his request. I chose to use a service called 2Captcha, due to its low cost and fast solving speeds. I followed their documentation[4] to implement my solution.

The user has to register an account on 2Captcha's website and set up his own API key. This key is inputted in the Settings section.

7.3.4 Other Anti-Scalping Measures

Another challenge that I encountered was a special kind of bot protection that was implemented on Buzzsneakers's checkout page. The developers included an extra field in their checkout form, where the customer has to solve a simple math problem as shown in Figure 7.4.

4. see `https://2captcha.com/2captcha-api#solving_captchas`

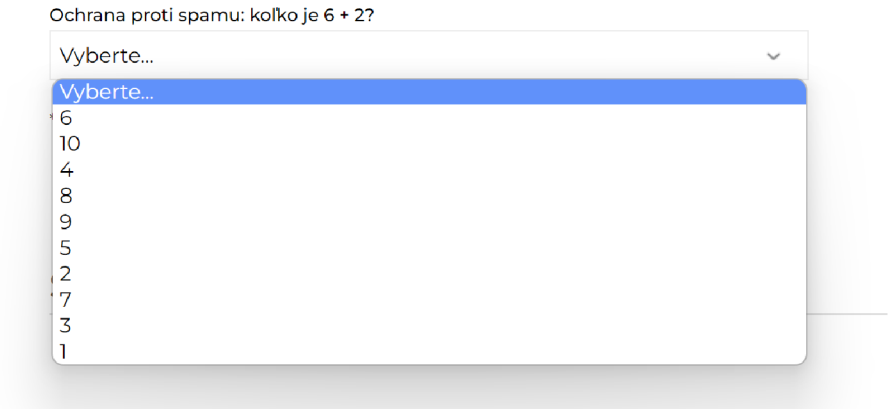

Figure 7.4: Anti-scalping measure at Buzzsneakers

This can be easily handled by scraping the form's page and extracting the math problem in a string format. Despite the fact, that the math problem is always different, it keeps the same format. I coded a function to parse the string of the math problem and calculate the result. The result is part of the post request that is sent to submit the checkout form, therefore I added this value to my post request's body.

7.4 Further Improvements

The application is designed to be easily expanded. This can be either by adding new store modules or new features. In this section, I am going to give thoughts about the future development of the application.

Sizing Selection

At the current stage, the application allows the user to select a size range for shoes. Despite that clothing is rarely targeted by scalpers, the ability to select a size range of clothes should be added to cover more products on each website. Furthermore, the application could provide the ability to select individual sizes instead of a size range.

Keywords Support

The current solution provides a checkout process automation for releases, in which the product page URL is known before. However, there are releases, in which the product link remains unknown until the product is released. The idea would be to add keywords support. This option would be an alternative to the product link input. The user could insert words, that match the expected product. During runtime, the module would search the website for those keywords until it finds a matching product.

On second thought, this feature comes with a lot of risks. The online release of Nike SB Dunk shoes hosted by Bonkers, a skate shop in Germany, in 2019 shows that this technique might not be the safest option. The store decided to combat scalper bots using the keywords technique and fooled them into purchasing photos of the sneakers instead of actual pairs. The store uploaded a product with the title "Pictures of shoes Nike SB Dunk" and wrote in the description of the product that the buyer shall not receive the shoes, but digital pictures of them. This kind of protection resulted in a negative reaction from scalpers, who purchased pictures of shoes for the price of the actual shoes [29].

Therefore, keywords support should be implemented very cautiously. A partial solution could be adding negative keywords to determine which words cannot be included in the product's title. However, it does not guarantee that the user would include every possible phrase for a trick product.

Captcha Solver Variants

Another improvement would be to add more captcha-solving providers, so the user could decide, which one he wants to use. At the moment, the application supports only 2Captcha, which forces the user to use this service.

Furthermore, the application could provide its own manual solver for captchas. The captchas would be sent to the user's interface and solved by the user. This way the user would not have to use a third-party service. However, this solver could be useful only when running

a small number of tasks and would require the constant presence of the user.

8 Conclusion

The main objective of the thesis was to explore the scalping area of web scraping. The analysis of legal and ethical background was carried out. The examination of current solutions showed that the most noticeable deficiencies were no control over the running application and the use of the command line interface. Based on the studied information a new web scraping application, that automates the checkout process on selected websites, was created.

The final form of the solution is a desktop application with a simple graphical user interface, that provides the ability to control its process during runtime, which makes the application unique among the existing solutions. The frame of the application is divided into the Tasks section, the Profiles section, the Proxies section, the Checkouts section, and the Settings section. The application is legal in every aspect and follows the ethical principles of web scraping.

The application is able to automatically purchase the desired item from all of the supported websites. The implemented websites are Queens, Popname, and Buzzsneakers. Some of them require solving bot protection challenges, which are handled by using a third-party solver. To avoid limiting the user's access to the website it was necessary to implement proxy support.

The application is built on an architecture that allows the addition of new store modules or control features that could be added in future development. The concept of suggested features is described at the end of the seventh chapter.

A Application Manual

The source code of the application can be found in the thesis archive in IS. The application is deployed locally and uses *pipenv* for dependency management. The following commands show how to run the application and execute tests:

```
Run the application:
1. Install pipenv if you do not have it already:
    pip install pipenv
2. Install all dependencies:
    pipenv install
3. Activate the Pipenv shell:
    pipenv shell
4. Start the application:
    pipenv run python .\main.py

Run tests:
    python -m unittest discover .\backend\tests\
```

A guidebook was created for users, who are not experienced in the scalping industry and might find some features unclear. It is available at `https://autocheckout-bot.gitbook.io/autocheckout-bot/` and consists of the introduction of sections in the user interface supplemented with screenshots of the application and setups for each store module.

Bibliography

1. ZHAO, Bo. Web scraping. *Encyclopedia of big data*. 2017, pp. 1–3. Available from DOI: https://doi.org/10.1007/978-3-319-32001-4_483-1.
2. SIRISURIYA, De S et al. A comparative study on web scraping. 2015. Available also from: http://ir.kdu.ac.lk/bitstream/handle/345/1051/com-059.pdf.
3. MITCHELL, Ryan. *Web scraping with Python: Collecting more data from the modern web*. " O'Reilly Media, Inc.", 2018. Available from DOI: https://dl.acm.org/doi/abs/10.5555/2904419.
4. GLEZ-PEÑA, Daniel; LOURENÇO, Anália; LÓPEZ-FERNÁNDEZ, Hugo; REBOIRO-JATO, Miguel; FDEZ-RIVEROLA, Florentino. Web scraping technologies in an API world. *Briefings in bioinformatics*. 2014, vol. 15, no. 5, pp. 788–797. Available from DOI: https://doi.org/10.1093/bib/bbt026.
5. VICENTE, Vann. What Are Online Scalpers and What Do They Do? 2021. Available also from: https://www.howtogeek.com/730574/what-are-online-scalpers-and-what-do-they-do/.
6. SHEIN, Esther. Global chip shortage: Everything you need to know. 2021. Available also from: https://www.techrepublic.com/article/global-chip-shortage-cheat-sheet/.
7. *Scalper Bots*. Netacea Ltd., 2020. Available also from: https://netacea.com/glossary/scalper-bots/.
8. COOPER, Stephen. 8 Best Bot Protection Software. 2022. Available also from: https://www.comparitech.com/net-admin/best-bot-protection-software/.
9. WELTY, Matt. Everything You Need to Know About Sneaker Bots. 2022. Available also from: https://www.complex.com/sneakers/how-to-use-sneaker-bots/.
10. DRAHOŠ, Jakub. *Webová aplikace pro online web scraping*. 2018. Available also from: http://hdl.handle.net/10467/83043. MA thesis. Fakulta informačných technológií ČVUT.

11. KROTOV, Vlad; SILVA, Leiser. Legality and ethics of web scraping. 2018.

12. TSE, Shing; BRYAN, Kristin L. hiQ Labs v. LinkedIn. 2022. Available also from: https://www.natlawreview.com/article/hiq-labs-v-linkedin.

13. O'ROURKE, Maureen A. Property rights and competition on the Internet: In search of an appropriate analogy. *Berkeley Technology Law Journal*. 2001. Available also from: https://scholarship.law.bu.edu/cgi/viewcontent.cgi?article=2530&context=faculty_scholarship.

14. *Trade secrets / regulatory data protection*. The United States Patent and Trademark Office, 2022. Available also from: https://www.uspto.gov/ip-policy/trade-secret-policy.

15. CONGER, Kate. Uber's Massive Scraping Program Collected Data About Competitors Around The World. 2017. Available also from: https://gizmodo.com/ubers-massive-scraping-program-collected-data-about-com-1820887947.

16. *General Data Protection Regulation (GDPR)*. Proton Technologies AG, 2022. Available also from: https://gdpr.eu/article-6-how-to-process-personal-data-legally/.

17. *sui generis*. Legal Information Institute, 2021. Available also from: https://www.law.cornell.edu/wex/sui_generis.

18. WOODHEAD, Gemma. Could Ryanair control use of its flight data by PR Aviation without database right? *Journal of Direct, Data and Digital Marketing Practice*. 2015, vol. 16, no. 3, pp. 234–235. Available from DOI: 10.1057/dddmp.2015.15.

19. MAZILU, Marius Cristian. Web Scraping and Ethics in Automated Data Collection. *EDUCATION, RESEARCH AND BUSINESS TECHNOLOGIES: Proceedings of 20th*. 2022, vol. 276, p. 285. Available from DOI: https://doi.org/10.1007/978-981-16-8866-9_24.

20. SUN, Yang; ZHUANG, Ziming; COUNCILL, Isaac G.; GILES, C. Lee. Determining Bias to Search Engines from Robots.txt. In: *IEEE/WIC/ACM International Conference on Web Intelligence (WI'07)*. 2007, pp. 149–155. Available from DOI: https://doi.org/10.1109/WI.2007.98.

21. BROUCKE, Seppe vanden; BAESENS, Bart. *Practical Web scraping for data science*. Springer, 2018. Available from DOI: https://doi.org/10.1007/978-1-4842-3582-9.

22. MENDOZA, Abner; CHINPRUTTHIWONG, Phakpoom; GU, Guofei. Uncovering HTTP header inconsistencies and the impact on desktop/mobile websites. In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 247–256. Available from DOI: https://doi.org/10.1145/3178876.3186091.

23. GOURLEY, David; TOTTY, Brian; SAYER, Marjorie; AGGARWAL, Anshu; REDDY, Sailu. *HTTP: the definitive guide*. " O'Reilly Media, Inc.", 2002.

24. LAWRENCE, Eric. *Debugging with Fiddler: The complete reference from the creator of the Fiddler Web Debugger*. 2012.

25. POLO, Guilherme. *PyGTK, PyQT, Tkinter and wxPython comparison*. 2017.

26. WOTTEN, Ian. The Best Python HTTP Clients For 2022. 2022. Available also from: https://www.scrapingbee.com/blog/best-python-http-clients/.

27. METWALLI, Sata. Python Databases 101: How to Choose a Database Library. 2021. Available also from: https://builtin.com/data-science/python-database.

28. *unittest — Unit testing framework*. The Python Software Foundation, 2022. Available also from: https://docs.python.org/3.10/library/unittest.html.

29. DESTEFANO, Mike. This Skate Shop Fooled Bots for the Parra x Nike SB Dunk Release. 2019. Available also from: https://solecollector.com/news/2019/08/this-skate-shop-fooled-bots-for-parra-nike-sb-dunk-release.