

Testing capabilities of bypassing anti-bot systems and extracting ecommerce data on a large scale.

A Thesis presented for the
Bachelor of Science
Degree
Saxion University of Applied Sciences

Sergiu Siminiuc
June 2022

Table of contents

1	Introduction	3
1.1	Topic and Context	3
1.2	Relevance and importance	4
2	Assignment.....	5
2.1	Planning.....	5
2.2	Problem Description.....	5
2.3	Core Assignment	6
2.4	Requirements list.....	6
2.4.1	Geizhals Crawler.....	6
2.4.2	Captcha Solver.....	7
3	Focus and Scope.....	8
3.1	Scope.....	8
4	Overview of the codebase structure	9
4.1	Base Platform	9
4.2	Process and data flow	10
4.3	Crawler	11
5	Research	12
5.1	Research questions	12
5.2	What is a Captcha?.....	14
5.2.1	Captcha Solving Solutions.....	15
5.3	Scrapers	16
5.4	Aims and Approach.....	17
5.5	Website research	17
5.6	Geizhal's captcha	20
5.7	ZennoPoster.....	22
5.8	Research conclusion.....	23
6	Development	24
6.1	Search action	24
6.1.1	URL crafting.....	25
6.1.2	Crawling by EAN	26
6.1.3	Solving the captcha.....	29
6.2	Ranking action.....	40
7	Conclusion.....	44
8	Bibliography.....	45

1 Introduction

1.1 Topic and Context

Priceintelligence GmbH is a Stuttgart based company that is providing software solutions for monitoring and analyzing online market data. (Priceintelligence GmbH, n.d.). Their goal is to gather and disseminate ecommerce data.

Data transparency for small businesses and large retails is at the core of their business. Priceintelligence crawls internationally through different marketplaces to collect data like product's prices, availability, ratings, reviews, descriptions, attributes and much more.

Initially I have started my job as an average Junior Software Engineer but quickly became interested in data collection and harvesting. I saw the business customers' interest and willing to pay for structured data, so I started to become more involved in the process of acquiring information, automatic the data flow to the clients and finding new techniques for intel aggregation.

Currently I am one of the responsible engineers for data crawling and API development and management at Priceintelligence GmbH. Also, I am developing and offering consulting services related to data repositories, medical data curation, *fairify-ing* (converting a dataset to a FAIR format (GO FAIR, n.d.)) and publication for nonprofit organizations at my own company.

1.2 Relevance and importance

Data is today's gold. The phrase may sound bold, but it is true. The human civilization advanced because of knowledge and data in general. Stored, published, written, spoken, or remembered, information always allowed us to progress forward. I want to mention an example at one of the foundations that contracts my company: Data processing for boys who suffer from Duchenne or Becker muscular dystrophies. By analyzing and crawling data from patients, clinical trials, and medical publications we can find connections between these and other diseases and provide affected families constant updates on the cure for the disease. I have given this example as it is more humane and understandable when it comes to the relevance of data in today's world.

Data is the lifeblood of our activities. Every day we speak, write, read the news, and make decisions and these activities are based on the existing data. From bad to good intentions, data affects every sector, and its importance cannot be denied.

The report is the result of my graduation project at Priceintelligence GmbH. It focuses on retrieving data from websites. The main task of this project turned out to be dealing with web pages that are protected by captcha. Captchas represent a service through which websites can protect themselves from automated bots, but this is also preventing ordinary people and companies from retrieving legit public data.

Chapter 2 will talk in more details about the assignment, its planning, description, and the list of requirements. Chapter 3 will go over the focus and scope of this project. Chapter 4 includes information about the research part of this project. Chapter 5 represents the development process.

2 Assignment

2.1 Planning

Initially, the project was thought of as a standalone project, but later into engineering too many dependencies from the base application were needed. The project proceeded to be implemented into the main application according to the application configuration policies. During the development, there was a need to switch the programming language from Python to Groovy.

All the conditions written in the Graduation Proposal are met except that the code is Groovy.

2.2 Problem Description

Big ecommerce companies with a vast catalogue of items are constantly in a fierce competition with their rivals. It is crucial for them to have the price of the items that they are selling on their competitors' websites. Article inventories can range from a couple hundred to millions.

The frequency of items' price updates ranges from daily to weekly. A human person could not look up every single item in the inventory, write down the data about it and finish them all in a day or a week.

Geizhals ecommerce platform is one of the biggest ones in Germany and Austria. It does not store physically any items that are present on its website, but it does offer free price comparison services. For over 2.4 million products, a user can see their offers on another third-party website all at once. (Geizhals, 2022) (ots.at, 2022). That is why Geizhals is a very demanded website from customers, and it is the reason this page was chosen for this assignment.

There is a need for a piece of software that can be integrated with the main platform that is able to lookup every single article, given a list of items. The code must be able to bypass any present obstacles to capture the data. The problem is that, currently, such a piece of software is not present, nor the website has been researched previously.

2.3 Core Assignment

The end goal of this assignment consists of getting to retrieve all the available information about a product, given a list of European Article Numbers. The process must be integrated flawlessly into the main platform.

2.4 Requirements list

1.1	Software must be compatible with the main platform.
1.2	Software must be able to parse and extract the correct attributes of products.
1.3	Software must be able to extract all offers from every single item that was found.
1.4	Software must be able to manage pagination. (Pagination represents a circumstance where information does not fit on a single page, thus it is spread across multiple web pages)
1.5	Software must be able to oversee network obstacles that prevent automation of crawlers (captchas, IP blocking etc..).
1.6	Software must be able to manage errors and log notable events.
1.7	Software must include tests for parsers.

2.4.1 Geizhals Crawler

A crawler for Geizhals ecommerce platform needs to be developed. It must be able to complete the following:

2.1	Craft URLs using EANs.
2.2	Solve any captcha that may be present.
2.3	Extract Product information and save it in the database connected to the base platform.
2.4	Extract all the products' offers information and save it in the database connected to the base platform.
2.5	Retain pages for potential future debugging.
2.6	Collect and store information about the crawling time, number of successful and failed attempts.

2.4.2 Captcha Solver

In case there is a captcha module present when crawling for products on Geizhals, it needs to be automatically solved. The tool must be able to bypass any captcha and retrieve the requested content. Solver is responsible for the following:

3.1	Recognize puzzles or images that are related to captcha.
3.2	Emulate human-like behavior.
3.3	Log the number of solving tries.
3.4	Use the same proxy as the crawler does.
3.5	Get to the wanted page.
3.6	Have a unique browser session for each solve.

3 Focus and Scope

The main goal of this project is to research and demonstrate the capability of *getting the data by any means necessary*. I want to present a peek of the raw data extraction and the challenges that come with it. It may sound intrusive, though I must mention that all the data is publicly available, and no laws are broken.

The research period consisted of four months of investigating, trial, and error. Investigating the websites, monitoring its requests, digging into applications, and playing a cat and mouse game with anti-bot systems.

3.1 Scope

The scope of the assignment is to create a crawler for the base platform and bypass any network obstacles. Picking CSS components and extracting information is not that difficult of a task, but bypassing blockers is. Solving a captcha or security system means free passes for all other websites that are using such a technique, not just Geizhals. These kinds of workarounds will provide customers the result – product data.

4 Overview of the codebase structure

The project is quite complex and required lots of dependencies and integration with the base platform. This chapter explains the structure of the base platforms and its crawlers in more detail.

4.1 Base Platform

The *Base Platform* that is referred through this document was originally called MI platforms and was developed jointly in late 2015 by a German software company RSMG which is now defunct and a small team of three independent developers. Later, the software was purchased by Parsionate GmbH, and a smaller child company (Priceintelligence GmbH) was created to operate and develop further.

The platform provided frame for crawlers to be developed. It contained all the necessary libraries, commonly used methods and test framework which allowed for a more standardized code structure for all future crawlers. It is responsible for running the crawlers, generating reports, saving data to the database, and providing API communication to the frontend module.

The frontend module is totally separate from the base platform. It communicates through API calls with the backend. The user interface provides useful functionalities for both the developers and customers. Developers can view running crawlers, analyze logs, import/export products, schedule future crawling actions and change customer's settings. Customers can view their imported products, crawled products' data & offers and statistics.

Overtime websites became more clogged, complex and the defense systems – more intelligent. This forced the developers to include more libraries, improve the proxy pool system and start using captcha solving APIs like 2captcha.

4.2 Process and data flow

The following describes the process of a crawler scraping for products. It represents a common, daily task within the company's operation.

1. The process for crawling starts with identifying the products to crawl. This is being done by mentioning the EANs or indicating the user for which all products should be crawled.
2. Validating the product is the second step. EAN standard and other parameters are checked for any flaws.
3. Search action. The product is searched on the indicated marketplace.
4. Optionally captcha needs to be solved.
5. Ranking action. The product's page is accessed, and data is being extracted.
6. Action is finished.

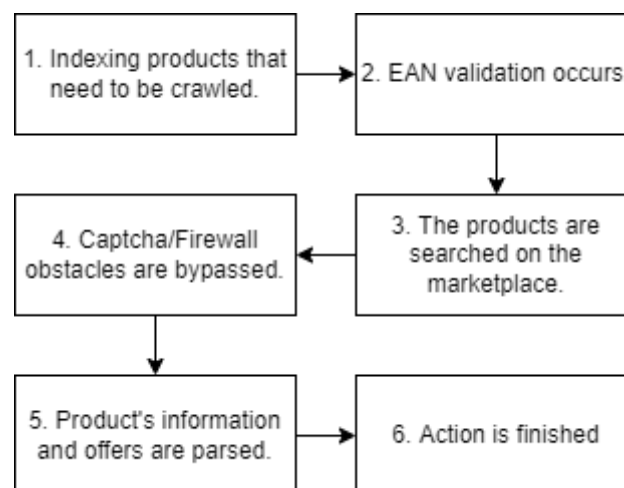


Figure 1 - Process and data flow for crawler actions

4.3 Crawler

By definition, a crawler is a tool that can extract data. At Priceintelligence a crawler is a piece of software usually written in the Groovy scripting language. It acts as a remote script that starts to crawl and populate the database with data. It can be triggered from the base platform and requires input parameters such as marketplace name, tenant key, EANs, EAN type, the option to save crawled content.

- Marketplace name, in this context, means the domain of the website that needs to be crawled.
- EANs is an optional parameter. It takes an array of EANs which match the products that need to be crawled. If left empty, it will crawl all available products of that user.
- EAN Type represents the type of an article's number. Some articles, like medicine products use PZNs instead of EANs. Other articles may use internal articles numbers. In this case we use Product Numbers (PNs).

The crawler itself can be configured according to the marketplace. Factors such as captchas, firewalls, incapsulated information, the way the information is stored in the web document (HTML elements or JSON), type of requests, headers. The scraper can be connected to third party software to forge certain requests. It can use various drivers, emulate all kinds of devices, and send requests through cellular, commercial, or private networks.

A parser is a key component of a crawler. It is responsible for converting DOM (Document Object Model) to organized and structured data that can be stored in the database. It can parse:

- Product's name
- Product's URL
- Product image's URL
- Price
- Original price & discount percentage (if there is a discount)
- Product's rating (usually one to five stars) and their total count
- Product's description
- Product's reviews and their total count
- Product's Brand
- Product offers' count
- Any other present attributes present

5 Research

5.1 Research questions

The research questions are taken directly from the Graduation Proposal.

How to incorporate a proxy manager into the crawler? A proxy manager is no longer required because it was decided to incorporate the crawler into the base application instead of developing it as a standalone project (Chapter 2.1)

How can I incorporate a captcha solver into the crawler? A captcha solver might be developed in accordance with the type of captcha that is used on the website. Usually, at Priceintelligence, 2captcha service is used to solve captchas. It is based on offshore human workers who solve captchas in large amounts. The service is one of the most renown in the world (ProWebScraper, 2022). The reason that it is widely used in most of the crawlers is the “plug and play” documentation that covers most of the modern programming languages. Another reason is the deep integration with the base platform. The library is so widely used and has been present for a long time in the project that it became a non-direct rule of thumb when dealing with captchas. In case the present defense mechanism is “out of the ordinary” then the use of ZennoPoster might come in handy. The choice of crawling solving solutions is compared and analyzed in chapter 5.2.

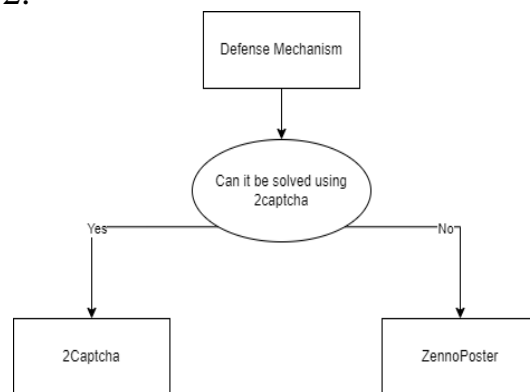


Figure 2 - Captcha Solving Solution Decision Diagram

How to efficiently parse the contents of the web page?

The efficiency of parsing a web page depends on the proxy load and the use of scraping libraries in the code. The system should be able to solve every request as fast as possible while using the least number of resources as possible. This can be achieved with good research on the available options and good coding. The proxy load refers to the amount of data that is used which represents the final service bill. To efficiently parse the contents of the webpage a good coding pattern, practice and thorough research is required.

Making good use of existing libraries such as JSoup, Jackson or Apache Common Tools makes the code much simpler and more readable. There is no need to re-invent the wheel for basic functions. In the context of scraping, having experience with CSS selectors is a huge plus. These are the patterns which are used to describe the HTML elements. Jackson is also a wonderful tool for mapping JSON objects.

A combination of practice, good coding patterns and knowledge are the key ingredients to an efficient parsing of the web content.

What would be a good way to save the data once it's retrieved? How will the database be designed?

The code is incorporated into the base application. This means that the only way to save the data is to use the base platform's MongoDB according to company's already pre-defined table entity scheme which is describe in more detail in Chapter 8 and 9 in the Technical Design (Appendix B). MongoDB uses JSON documents to store the data, which is very convenient as some of the websites offer product's data in this format. The preferred method to do it is to use the MongoDB JDBC driver and use the functions available in the base platform's interface (Spring interface code).

How will the crawler be deployed? As written in Chapter 2.1, the application will be incorporated into the base platform instead of being deployed as a standalone application. The code will be integrated using the pre-defined libraries and coding style of other already-developed crawlers. Another option would be to make the crawler a standalone application (original plan), but due to the large number of dependencies from the main platform it was decided to integrate to the base platform.

How will the crawler communicate with the main platform?

The code will be incorporated into the main application meaning it will make use of the base platform's API calls. As mentioned in the previous research question, another option would be to make the crawler independent, but plans have changed,

5.2 What is a Captcha?

Originally captchas did represent computer-generated images or just plain distorted text (Ahn, 2017). The “reCAPTCHA” project took off in late 2000’s as website bots became more advanced and damaging. It was a breakthrough moment as a solution for stopping automated bots was finally found. Since then, it is a cat and mouse game between engineers of both sides. While the intent of captchas is honorable, sometimes it just makes the browsing experience worse for the average web explorer. Over time, captchas got more advanced and made the solving of them more difficult.

According to (Built With, n.d.), Google’s reCAPTCHA accounts for about 91% of the captcha market share. It is followed by hCaptcha and then other less-known captchas. Most of the captchas that are encountered while engineering solutions for big ecommerce website crawling reCAPTCHA, hCaptcha and text-based captchas. Sometimes computing processing power is used as a PoW (Proof of Work) for the website to verify if the request is legitimate or not.

5.2.1 Captcha Solving Solutions

There are some captchas solving options, depending on the complexity of the problem. As mentioned in chapter 5.1, there are some SaaS (Software as a Service) solutions. Out of all the popular service providers, I found 2captcha the cheapest and best documented. The API was easy to integrate with the base platform. Below are compared 2captcha.com and two other non-SaaS methods of solving the captcha.

I have included ZennoPoster as there is simply no other alternative software like this on the market today. There are applications like OctoParse or ParseHub, but those do not offer the necessary level of configuration that I was looking for.

ZennoPoster satisfies my requirements of:

- Communicating with the base platform
- Communicating with the database
- Run custom code

Options	Advantages	Disadvantages
2captcha.com	<ul style="list-style-type: none">• Easy implementation (Plug and Play) solution• Scalable• Well documented	
Mobile app sniffing	<ul style="list-style-type: none">• Easy data extraction• Mobile request needs less data which means less money spent on proxies	<ul style="list-style-type: none">• Takes time to investigate with an average of 5-10% success rate
ZennoPoster	<ul style="list-style-type: none">• Configurable• Scalable (depends on the on-premises servers)• Documented	<ul style="list-style-type: none">• License is expensive• Takes lot of time to implement but has an average 80% that it could solve the captcha

Table 1 - Advantages and Disadvantages of captcha solving options

5.3 Scrapers

Web scraping is used to extract data from websites in an automated way. It represents a piece of code that has the task to retrieve information, and most of the times, simulate a human being.

The usage of web scrapers can vary on a large scale. In ticket sales, advanced bots are usually tasked with sniping the spot as soon as it goes live. This activity is in the gray area as the perpetrators usually list the tickets later for a larger amount and it prevents the average customer from enjoying the experience at said event.

At the base principle, web scrapers are usually used to automate routine tasks on the internet. Filling some hours in a sheet, checking for flights are filling in forms can be easily automated with the right libraries.

The biggest perk is that it saves us lots of time and energy, but it makes the internet feel less human sometimes.

According to (Lalani, 2021) the most popular Java libraries for web scraping are Heritrix, Web-Harvest, Apache Nutch, Jaunt, StormCrawler, Gecco, WebSphinx, JSoup, HTMLUnit and Norconex. Heritrix is more focused towards web archiving. Apache Nutch's documentation at the time of base platform's project start (2015) was not thorough thus it was not chosen. For my research, JSoup and HTMLUnit were the libraries of choice mostly because I already have lots of experience with them, the documentation is good, and I do not have to import new code to the base platform. Also, I have encountered issues using other libraries with the base platform's Spring version: application was slow, incompatibility bugs between the browser drivers and libraries, testing documentation was not straightforward. Those are the reasons that I stuck with JSoup and HTMLUnit.

5.4 Aims and Approach

The main goal is to extract data and metadata from the website.

- Website's layout needs to be understood. CSS selectors and network packets should be analyzed. The approach is to compare the html and XHR packets content. The best-case scenario is to use API requests to retrieve JSON objects containing product and offer data, but usually it is not the case.
- Look out for captcha and ban pages. The approach is to use request platforms such as Postman, inject a proxy from a pool that will be used in the future and spam certain pages while deleting all cookies and cache in between.
- Solve captcha pages in case there are any. The current approach is the following: try 2captcha service then sniff mobile application network packets and last – implement ZennoPoster solution.
- Develop a proxy pool management which works with common and domain divided pools. The approach is to implement a Spring Framework component with the required functions.

5.5 Website research

Before any crawler development, a thorough research about the website must be done. An ecommerce website is quite a complex and rich cluster of data that needs to be picked apart. Anything from html page layouts, JavaScript code, API requests, server-side processes need to be analyzed. The goal here is to use as minimum resources as possible to get the necessary data. A JSON file would be much more efficient than a whole html page as it cuts down on proxies' costs.

It starts with a browser with network development tools and an API platform. For current project Chrome and Postman were used. An initial overview of the website is done to comprehend the structure.

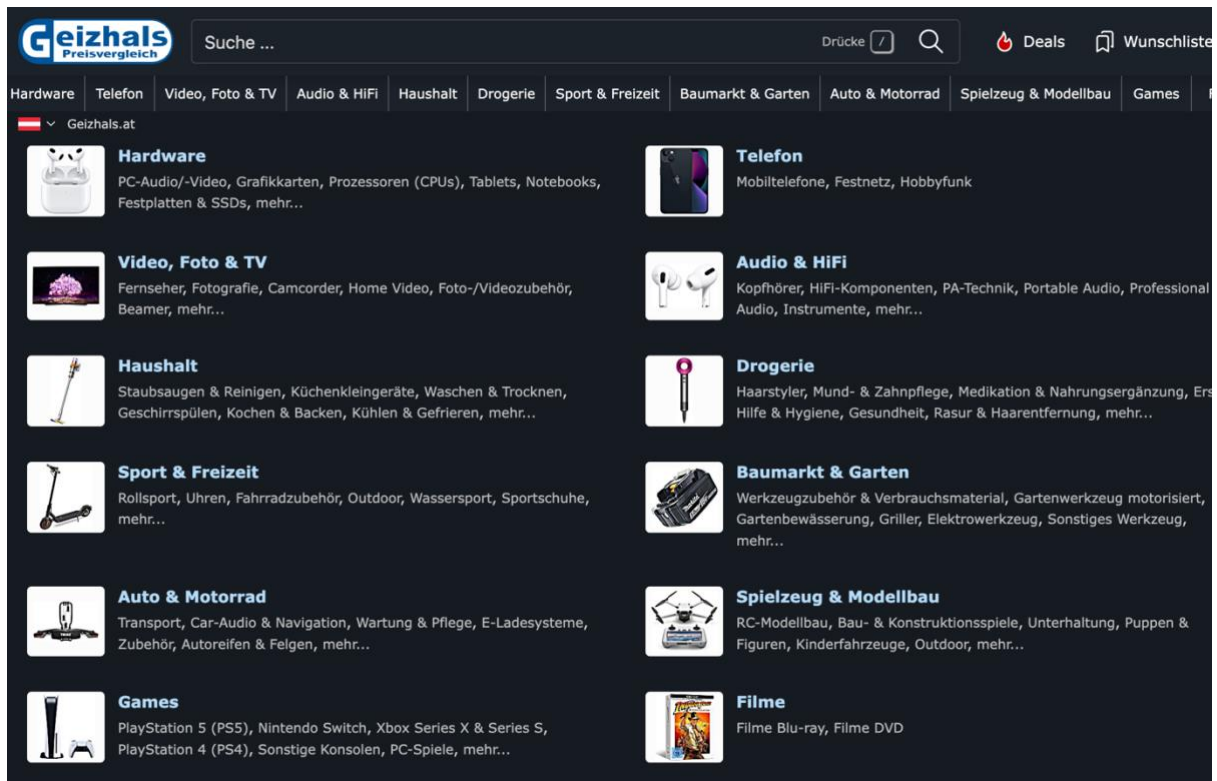


Figure 3 - Geizhals.at front page User Interface

Geizhals.at and Geizhals.de have similar layouts. It comprises a top search bar with a list of categories underneath. The product's page is filled with useful information. There are basic product attributes like title, description, and specifications, image link. This is the mandatory, minimal data that is needed to be able to store the product in the database.



Figure 4 - Geizhals.at Product page User Interface

A great feature of Geizhals is that it contains the offers from other ecommerce website for this product. It acts like a sort of an aggregator. Instead of searching the same item on multiple marketplaces and develop a separate crawl for each, time can be spent developing a crawler for this platform and collect data on

multiple other marketplaces. The mandatory data that we need to save the product's offers is:

1. The offer's name
2. Price
3. Shipping price
4. Payment options
5. Delivery time
6. Description
7. Offer's URL

Information like reviews, customer's comments, ratings, and attributes are welcome too.

Preis exkl. Versand*	Anbieter	Händlerbewertung	Verfügbarkeit / Versand*	Produktbezeichnung des Händlers
€ 799,00 zum Angebot	dshop.at Infos AGB	★★★★★ 4.7 33 Bewertungen	siehe Shop kein Versand. Abholung gegen Barzahlung möglich (A-1100 Wien, A-1160 Wien, A-2230 Gänserndorf)	Apple iPhone 13 128GB Mitternacht Preise und Reservierungen sind nur nach erfolgreicher E-Mail-Bestätigung gültig! AKTIONSPREIS!!! GRATIS für Sie dazu ein PANZERGLAS!!! Preis vom: 25.05.2022, 18:39:03 (Preis kann jetzt höher sein!)
€ 825,00 zum Angebot	mobizone Infos AGB	★★★★★ 5.0 159 Bewertungen	Elisale Wien 22: lagernd/abholbereit Elisale Wien 6: lagernd/abholbereit Stand: 28.05.2022, 16:45 Kein Versand möglich. Abholung gegen Barzahlung möglich (A-1060 Wien, A-1220 Wien)	Apple iPhone 13 128GB Mitternacht + gratis Panzer Glas Neu und original verpackt Preis vom: 28.05.2022, 17:15:11 (Preis kann jetzt höher sein!)
€ 826,88 zum Angebot	amazon.de Amazon.at Hinweis: Firmensitz in Deutschland Infos AGB	★★★★★ 2.8 2894 Bewertungen	Noch nicht lieferbar Kreditkarte, Lastschrift GRATISVERSAND.	Apple MUPF32D/A Apple iPhone 13 (128 GB) - Mitternacht Preis vom: 28.05.2022, 17:28:56 (Preis kann jetzt höher sein!)
€ 829,00 zum Angebot	Topprodukte Media Markt.at Infos AGB	★★★★★ 4.7 4976 Bewertungen	Online Shop: lagernd, Lieferung in 3-5 Tagen Elisale: In allen MediaMarkten bestellbar (3-4 Werktage) Stand: 28.05.2022, 17:25 Kreditkarte, PayPal, sofortüberweisung.de € 2,99. Abholung und Preisüberprüfung	APPLE MUPF32D/A APPLE iPhone 13 128GB Mitternacht; Smartphone (Art# 1858721) iPhone, USB-C auf Lightning Kabel, Dokumentation (Abbildungen: MagSafe Zubehör ist nicht im Lieferumfang enthalten) Preise gelten bei Bestellung im Mediamarkt Onlineshop! Preis vom: 28.05.2022, 17:21:31 (Preis kann jetzt höher sein!)

Figure 5 - Geizhals.at Product's Offers Page User Interface

Back to our original goal, crawling data from EANs. First thing to do is to input the EAN in the search box. As an example, we can use 194252708118 which is the article code for Apple's iPhone 13 128GB. Upon pasting the number in the search field and hitting enter, we encounter the following page:

Figure 6 - Geizhals.at Captcha Puzzle Page

This is a captcha page. Specifically, Friendly Captcha. Some ecommerce websites implement such techniques to deter bots and crawlers from massively collecting their data. Although some ecommerce platforms offer public APIs for developers to extract data more easily, some hire external companies like Data Dome and Cloudflare to prevent crawling. Most of the platforms who use captchas deploy operate Google’s reCAPTCHA, hCAPTCHA or distorted text like Amazon’s captcha:

Usually, third party services such as 2captcha is used to solve them through bots, but at the time of writing this document Friendly Captcha is not supported.

After solving the captcha, the website redirects to the product’s page. An analysis of Fetch/XHR networks shows that there are no JSON responses that we could use to bypass the captcha or process the data faster.



Figure 7 - Chrome DevTools Network Fetch/XHR packets for geizhals.at product page

The conclusions of the website research are the following:

- There is a need to develop a captcha bypass mechanism. There are no other quick ways to bypass it. Friendly Captcha is a new type of captcha that needs to be analyzed thoroughly.
- There is a need to create a parser for the product page using CSS selectors, which are patterns used to select the elements (W3 Schools, n.d.) that contain the desired data.

5.6 Geizhal’s captcha

Upon researching the website, I found out that Geizhals is using FriendlyCaptcha. It a proof-of-work based solution in which the user’s device does all the work (Friendly Captcha, n.d.). It is not a text-based captcha or some kind of puzzle. It does some computing work on the device and then marks the captcha as solved. I found out that running this captcha continuously takes longer and longer. I reached a 10-minute wait at some point. This shows that a new browser session must be initiated before each captcha solving.

The inner javascript code was obfuscated and after spending about three days of analyzing and trying to de-obfuscate it, I came up with no results. The captcha required Javascript which meant the use of third-party SaaS captcha solutions was not a viable choice as none of them support this new type of captcha.

An analysis of the JavaScript code reveals that the Captcha's puzzle solving occurs on the user's machine. Sometimes solving the puzzle can be done by injecting JavaScript code into the crawler's script. In this case the code is heavily obfuscated, and it would take a lot of time to decode it.

A good reason for not spending research time on de-obfuscating JavaScript code or inspecting mobile applications' network packets is that JavaScript code or the encryption key can be often changed, thus making the crawler's availability time very limited. Friendly Captcha has a whole team dedicated to preventing someone using their source code. Likewise, Geizhals has a security team ensuring that their application communication is secure and cannot be exploited.

Geizhals also has a mobile app. It was downloaded on a phone and a library called "Man in the middle proxy" (mitmproxy) was used to intercept the communication. The result showed that the app used an encryption key. By unpacking the .apk file for Android's Geizhal app, the encryption key could not be retrieved. Although the encryption key could probably be uncovered by doing advanced security investigations on the mobile applications, it would require more time spent.

I was left with the last choice: ZennoPoster. I have noticed that it uses quite a lot of computing power which meant that powerful servers were needed in order to solve these captchas in parallel.

5.7 ZennoPoster

ZennoPoster is a software product developed by ZennoLab. It is a solution for automating tasks on the internet. It can execute projects by tens and hundreds of threads simultaneously.

The features are highly customizable. It makes use of any kind of proxies, internal, external, and global variables, database connectivity, all modern browser agents and much more.

At Priceintelligence, ZennoPoster is the last option for captcha solving. While there are good external services such as 2captcha.com that deal with popular captchas, there are some that cannot yet be solved by them. The solution for this is to create a custom “captcha-solver” for the specific marketplace. It takes much more time to develop and test it. Website changes may affect the script, but in the end the goal must be met – retrieve the data by any means.

Friendly Captcha markets itself as a service to protect organizations and small businesses. It is a new service and searching for bypasses on internet forums does not reveal anything at all. The service is even used by European Commission, Zalando and Red Cross. The goal here is to bypass it by simulating the user’s actions and make the connectivity with our main application.

5.8 Research conclusion

In conclusion, the base platform's libraries of choice will be used for Geizhals crawler development. Having experience with said libraries, lots of documentation and code crawler code examples at my disposal makes the use of these choices clearer. In case there are captchas present, 2Captcha will be the service of choice, followed by ZennoPoster.

The choices are the result of an extensive research for possible solutions. Parsing libraries for Groovy were analyzed and compared in chapter 5.3 and Captcha solving solutions were analyzed and compared in chapter 5.2. I found those solutions to work with the technology stack that the base platform and my crawler are based upon.

I dove deep into the documentation of these solutions and did lots of trial-and-error actions. For parsing libraries, I have created tests (which are included in the uploaded files) that make use of said dependencies. For captcha solving solutions I have looked mostly at the price, speed, documentation and integration capabilities.

6 Development

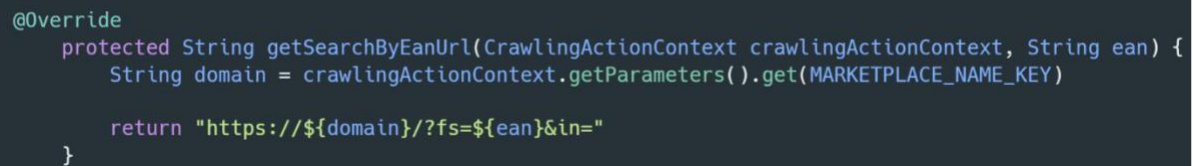
6.1 Search action

The search action is the first crawling event after a product has been validated in the database. After a company has imported its products, those are automatically checked for any errors like missing characters, wrong EAN length, missing fields etc.

Search is an optional action. If the product has been previously crawled and there is a listing URL present in the database, the application may skip searching and proceed straight to the ranking action. It depends on the TTL (Time to Live) property.

Extending the base abstract actions from the MI module, an action is created *GeizhalsSearchCrawlingAction*. While most functions can be left by default there are some that need a custom implementation. Given an EAN, the application should be able to craft an URL through which it can search for the product.

6.1.1 URL crafting



```
@Override
protected String getSearchByEanUrl(CrawlingActionContext crawlingActionContext, String ean) {
    String domain = crawlingActionContext.getParameters().get(MARKETPLACE_NAME_KEY)

    return "https://${domain}?fs=${ean}&in="
}
```

Figure 8 - Search Action URL Crafting

Domain is the marketplace of the crawler. In this case *geizhals.at* would be used as the domain. Following the research that we did for the marketplace it was concluded that by replacing the *?fs* parameter we can insert any valid 13-digit EAN.

6.1.2 Crawling by EAN

After crafting the URL by replacing the parameters, the request itself must be made. The function will retrieve the crafted URL, sleep for a random amount of time (between 3 and 5 seconds) then send the request. After the request has successfully been sent, the response content may be saved. The option to save the content is indicated in the platform's user interface whenever the action is started. If option is active, the content will be saved in the machine's file directory. This is mostly useful for debugging. By saving the crawled content the developer can quickly run tests for it and understand whether the content format has been changed. This is also another reason why JSON responses are preferred over HTML ones. The page of a big ecommerce website is often changed, and every modification is time spent by the developer. Finally, the parser service is called and applied to the content that was retrieved.

```
@Override
protected BaseSearchResultParserResult doCrawlByEan(final CrawlingActionContext crawlingActionContext, final
String ean, final String marketplaceName, final IProduct product) {
    String url = getSearchByEanUrl(crawlingActionContext, ean)

    doDelay(this.delayMin, this.delayRandomPart)

    GeizhalsUtils.sendRequest(crawlingActionContext, url)
    CrawlUtils.saveCrawledContent(product.getEan(), saveCrawlingContentPath, crawlingActionContext)

    return crawlingActionContext.getParsingService().parse(
        new GeizhalsSearchResultsParser(), crawlingActionContext.getCrawlingDriver().getCrawledContents())
}
```

Figure 9 - doCrawlByEan function

Since the search of a 13-digit EAN on *geizhals.at* is met with Friendly Captcha, the system makes use of the Zenno Poster service.

```
static void sendRequest(final CrawlingActionContext context, final String url){
    if(!imitateRequest(context, url)){
        throw new ActionException("Error could not simulate request through ZP !!!")
    }
}

static boolean imitateRequest(final CrawlingActionContext crawlingActionContext, @NotNull final String url){
    boolean imitated = crawlingActionContext.getCrawlingDriver().getZennoPosterService()
        .imitateRequest(crawlingActionContext, crawlingActionContext.getProxy(), url, false, 1)
    CrawlUtils.saveCrawledContent("ffg", CONTENT_PATH, crawlingActionContext)
    return imitated
}
```

Figure 10 - sendRequest function

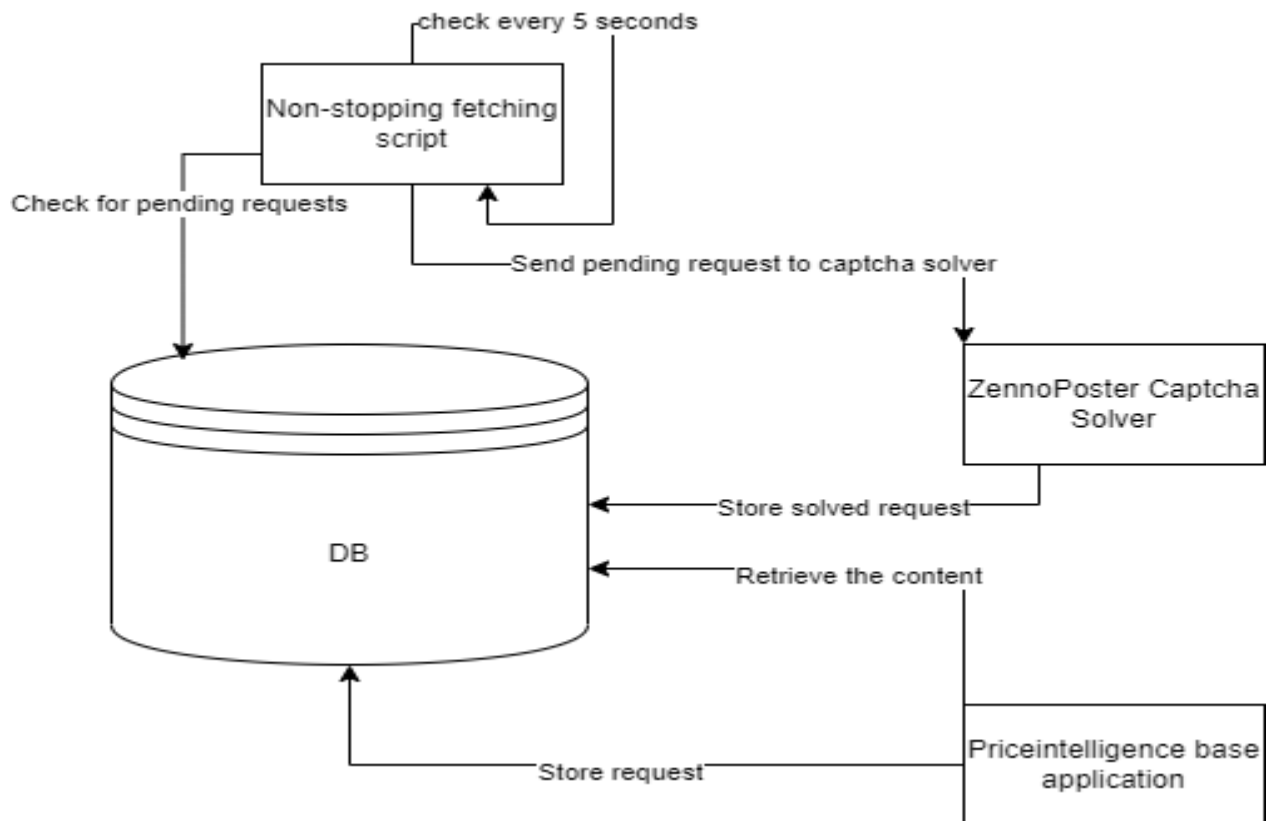


Figure 11 - ZennoPoster, Database and Main app connectivity diagram

Figure 11 represents the workflow of the ZennoPoster whenever it received requests from the base platform.

1. The Priceintelligence base application will store a request object in the database.

```

1 {
2   "_id" : ObjectId("629255dc2be26d92e255385d"),
3   "proxy" : "",
4   "request" : "https://geizhals.at/?fs=4016803055259&in=",
5   "marketplace" : "geizhals.at",
6   "status" : "CREATED",
7   "creationTime" : ISODate("2022-05-28T17:03:24.615+0000"),
8   "params" : {
9     "proxy" : "",
10    "pages" : "1",
11    "method" : "simulation",
12    "marketplace" : "geizhals.at",
13    "pageurl" : "https://geizhals.at/?fs=4016803055259&in=",
14    "key" : "5f82e297b51108cc0d4b979de50ee0355f49cfc466a3a6e7a0e4af9b46a79669"
15  },
16   "_class" : "de.pi.capmonsterbalancer.models.CrawlingTask"
17 }
  
```

Figure 12 - JSON request object

2. It will contain the proxy that was meant to be used, the crafted URL, marketplace name, status (at the time of request creation is CREATED), creation time and parameters including the number of pages and key. The proxy will be used later for human page view simulation and captcha solving. The key is used globally to identify the captcha solving task. Using this key, the application will retrieve the content and parse it. The webpage may be composed of multiple pages; thus, the number of pages is included.
3. A script will run in loop every 5 seconds. It will check the database for any task labeled as CREATED in the status field. Once it has found one it will mark the request as PENDING and go to the next step.
4. The never-ending script will summon an instance of the ZennoPoster captcha solving custom application and give all the necessary information to it.
5. The ZennoPoster instance will retrieve the task and start solving it. Once it has solved the captcha and retrieved the content, it will modify the object in the database, adding the content and changing its status to SOLVED.
6. Finally, the base application can retrieve the content and start parsing it.

6.1.3 Solving the captcha

For ZennoPoster to communicate with the application, the MongoDB database will be used to synchronize actions and requests' results. The database will be hosted on a separate server. It will take and store requests made by the base platform. A ZennoPoster script will be responsible for fetching for new requests every 5 seconds and proceed to solve them.

Setting up the new project, we will start with the profile. This will generate a fingerprint specific for the set person.

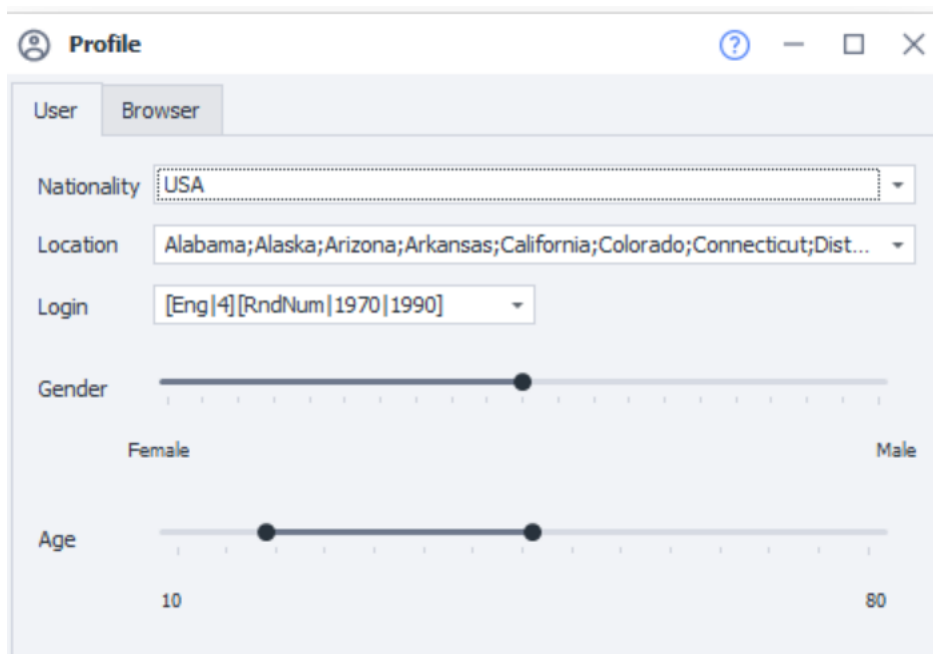
The image shows a software window titled "Profile" with a user icon and standard window controls (help, minimize, maximize, close). Inside the window, there are two tabs: "User" and "Browser", with "User" currently selected. Below the tabs, there are four configuration fields: 1. "Nationality" is a dropdown menu set to "USA". 2. "Location" is a dropdown menu showing a list of US states: "Alabama;Alaska;Arizona;Arkansas;California;Colorado;Connecticut;Dist...". 3. "Login" is a dropdown menu set to "[Eng|4][RndNum|1970|1990]". 4. "Gender" is a horizontal slider ranging from "Female" on the left to "Male" on the right, with a black dot indicating the current selection is slightly past the midpoint. 5. "Age" is a horizontal slider ranging from "10" on the left to "80" on the right, with two black dots indicating a range from approximately 15 to 55.

Figure 13 - ZennoPoster profile data

Nationality and location are US based even though these can be changed to any other ones. It is important though that the selected zone can browse the marketplace. There are instances in which companies geo-block certain regions.

Login represents the spoken language and the age of the user. Ideally, we would want a balanced mix between male and female profiles, in their adulthood.

Variables			
Custom Auto Environment Global Json Xml			
	Name	Value	Default value
→	abschickenText		
	captchaStatus		
	connect		
	collection	crawlingTask	
	document		
	taskId		
	taskStatus		
	userDB	readwrite	
	passDB	captchaReadWrite	
	dbName	capmonsterbalancer	
	dbHost	168.119.5.57	
	dbPort	27017	
	taskProxy		
	proxy		168.119.5.57:25216

Figure 14 - ZennoPoster variables table

Project variables are set according to the website's needs. There are abstract fields such as a database's credentials, proxy, task id and status etc.

AbschickenText is added to save a specific variable that is needed in Friendly Captcha's solving. Proxy has a default value representing a gateway to retrieve dynamic proxies in case our proxies are not present.

Variables can be both global and internal only. Global variables can be used across other projects for easier monitoring and change of fields. Each variable must be configured and added manually.

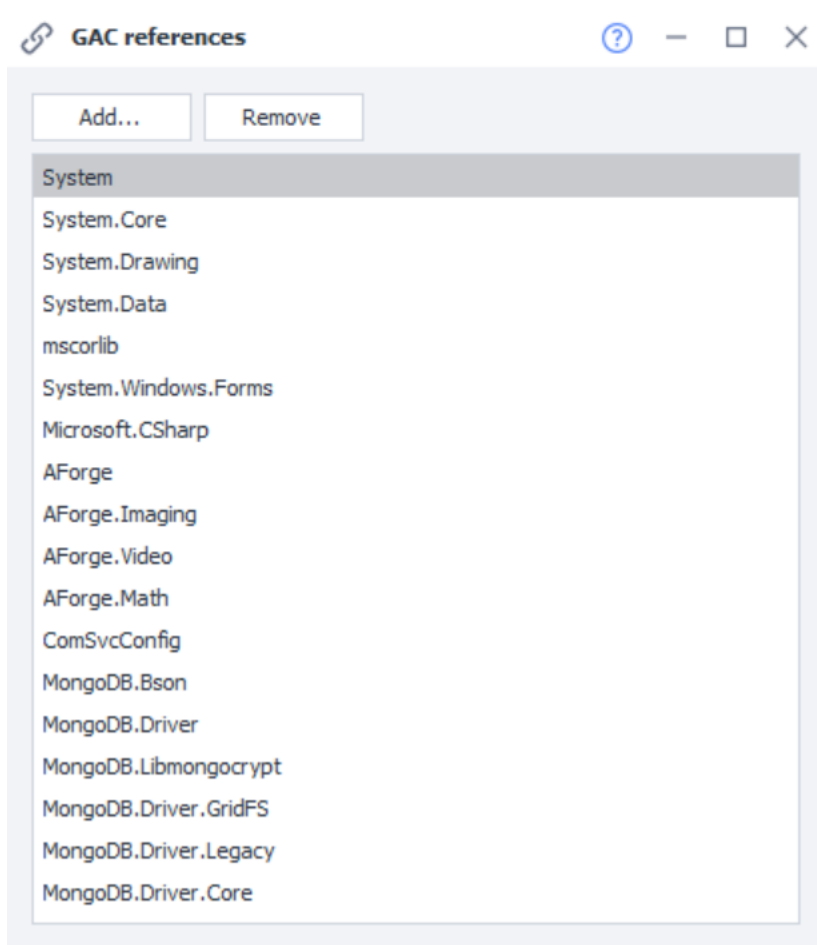


Figure 15 - ZennoPoster GAC References

GAC references are libraries used for the ZennoPoster project. These are .dll files that perform specific tasks. In this project, MongoDB drivers were added to make the connectivity with the database. Other libraries can also be included to solve imaging puzzles or render 3D objects. These files will allow us to call functions in our C# custom scripts.

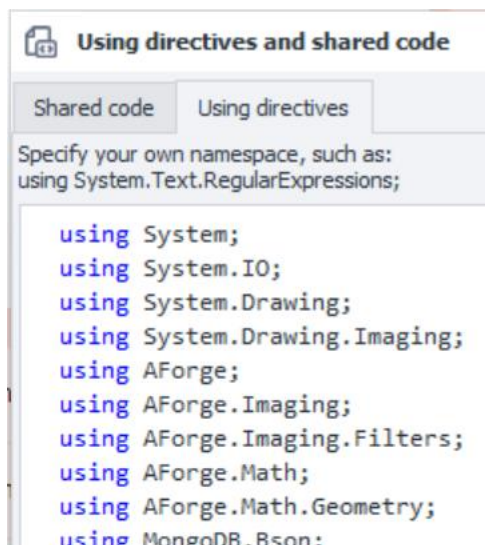


Figure 16 - ZennoPoster directives and shared code

Directives are imports done to be able to call the driver's functions.

Code in ZennoPoster is written in C# only and it can make use of libraries and change internal settings.

Using this function, we retrieve the database name that we previously mentioned in our project's variables.

```
public static IMongoDatabase getConnection(IZennoPosterProjectModel project)
{
    var dbName = project.Variables["dbName"].Value;
    IMongoDatabase database1 = null;
    try{
        database1 = DBClient.GetDatabase(dbName);
    } catch (Exception e){
        setClient(project);
        database1 = DBClient.GetDatabase(dbName);
    }
    return database1;
}
```

Figure 17 - ZennoPoster C# MongoDB connectivity code

After the connection with the database is set, the project can make use of other credentials and variables.

```
private static void setClient(IZennoPosterProjectModel project)
{
    //var connectionString1 = "mongodb://readwrite:captchaReadWrite@localhost:27017";
    var passMongo = project.Variables["passDB"].Value;
    var userMongo = project.Variables["userDB"].Value;
    var dbHost = project.Variables["dbHost"].Value;
    var dbName = project.Variables["dbName"].Value;
    var dbPort = Convert.ToInt16(project.Variables["dbPort"].Value);
    var credentials = MongoCredential.CreateCredential(
        databaseName: dbName,
        username: userMongo,
        password: passMongo
    );
    var settings2 = new MongoClientSettings
    {
        Credential = credentials,
        Server = new MongoServerAddress(dbHost, dbPort)
    };
    DBClient = new MongoClient(settings2);
}
```

Figure 18 - ZennoPoster C# MongoDB connectivity code

All this code is placed in the “shared code” field. It allows for an easier access by all other projects to the database connectivity modules.

Everything is set to develop the captcha solving mechanism.

```
var collectionName = project.Variables["collection"].Value;
var dbConn = ZennoLab.OwnCode.MongoConnection.getConnection(project);
var collection1 = dbConn.GetCollection<BsonDocument> (collectionName);

var marketplace = Builders<BsonDocument>.Filter.Eq("marketplace", "geizhals.at");
var status = Builders<BsonDocument>.Filter.Eq("status", "CREATED");
var combineFilters = Builders<BsonDocument>.Filter.And(marketplace, status);
//IMongoCollection<BsonDocument> zzz = collection1.Find(filter)
lock(SyncObject) {
    var doc = collection1.Find(combineFilters).FirstOrDefault();
    //var doc = collection1.Find(combineFilters).ToList();
    //var docNr = collection1.Find(filter).FirstOrDefault();
    if(doc != null){
        //project.SendInfoToLog("" + doc.ToString(), true);
        project.SendInfoToLog("" + doc.ToJson(), true);
        project.Variables["document"].Value = doc.ToJson();
        string pattern = "(\\ )*(?<=\\_id\\\"\\\\ :\\\\ ObjectId\\\\\\\\(\\\").*?(?=\\\"),\\\\\\\\ \\\\\"\\\\\\\\)\"";
        Regex rg = new Regex(pattern);
        project.Variables["taskId"].Value = rg.Matches(project.Variables["document"].Value)[0].Value;
        if(project.Variables["taskId"].Value != ""){
            var filter = Builders<BsonDocument>.Filter.Eq("_id", new ObjectId(project.Variables["taskId"].Value))
            var update = Builders<BsonDocument>.Update
                .Set("status", "PENDING");
            collection1.UpdateOne(filter, update);
            project.Variables["taskStatus"].Value = "PENDING";
        } else {
            project.Variables["taskStatus"].Value = "CREATED";
        }
    } else {
        project.Variables["document"].Value = null;
        //project.SendInfoToLog("There are no CREATED tasks in db !", true);
    }
}
```

Figure 19 - ZennoPoster C# code

First the collection is retrieved from the project’s array of variables. The database connection is taken from the shared code directory where the GAC references, drivers and connectivity code was developed.

Next, the application needs to be able to filter tasks related to geizhals.at. The database is being used by other big marketplaces and it is important to be able to distinguish the correct tasks. In this case, tasks with geizhals.at for marketplace name and CREATED for status are being retrieved. Logs must be saved for easier debugging and future analytics. Metadata like time to perform these solving tasks can be tracked and monitored. Custom scripts can be set up to recognize any spike in the solving time thus addressing the issue faster. Regex patterns are used to retrieve certain properties of the task object such as the task id.

After retrieving the request task object, an additional check is needed. Here, an internal feature of ZennoPoster is used to verify the integrity of the document variable value.

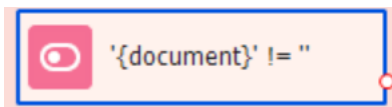


Figure 20 - ZennoPoster comparison module

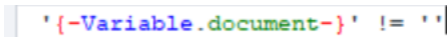


Figure 21 - ZennoPoster comparison module

A few regex requests are made to extract the proxy from the task. The reason for using the regex expressions in this and the previous task is because the drivers do not have the capability yet to map JSON fields to strings or variables. I found regex to work well as a workaround.

After further validation, the solving process can be started. In case there are any errors encountered until now the application will log an error and metadata for future quick fixes. If everything is well and variables check out a browser instance is summoned. Accordingly, it is advised to customize the instance based on the website environment and generally, variables that were found to work in our favor.

Clearing cookies, cache, enabling features and parameters can reduce the data used. As mentioned, it is crucial to save resources. It may not seem much, but every kilobyte can mean megabytes or even gigabytes whenever we talk about performing tens of thousands of these requests. As a precaution, again, a proxy gateway is inserted in case the normal proxies fail.

```
instance.ClearCookie();
instance.ClearCache();
instance.SetBrowserPreference("media.peerconnection.enabled", false);
instance.UsePlugins = false;
project.Variables["startTime"].Value = (DateTime.UtcNow - new DateTime(1970, 1, 1)).TotalMilliseconds.ToString();
var taskProxy = project.Variables["taskProxy"].Value;
//instance.ClearProxy();
string activeProxy = instance.GetProxy();
if(activeProxy == null || activeProxy == ""){
    if(taskProxy == null || taskProxy == ""){
        taskProxy = project.Variables["proxy"].Value;
        if(taskProxy.Contains("1.1.1.1")){
            taskProxy = taskProxy.Replace("1.1.1.1", "ctr-2-30m.geosurf.io");
        }
    } else {
        project.Variables["proxy"].Value = taskProxy;
    }
}
instance.SetProxy(taskProxy);
}
```

Figure 22 - ZennoPoster browser client initialisation instance code

The URL variable is then used to access the webpage. A pause of two to four seconds is in place. Next, the page is loaded and using a xPath it is possible to determine if the captcha modules were loaded and ready to be used.

“*Hier klicken*” is checked by following the xPath:

“/html/body/div[1]/div[2]/div/form/div/div/div/button”

“*Hier klicken*” is saved temporarily to project variables as captcha status. It will tell us whether the captcha will be solved in the future this allowing the application to access the product’s ranking page.

Next a click is simulated to proceed with captcha solving.

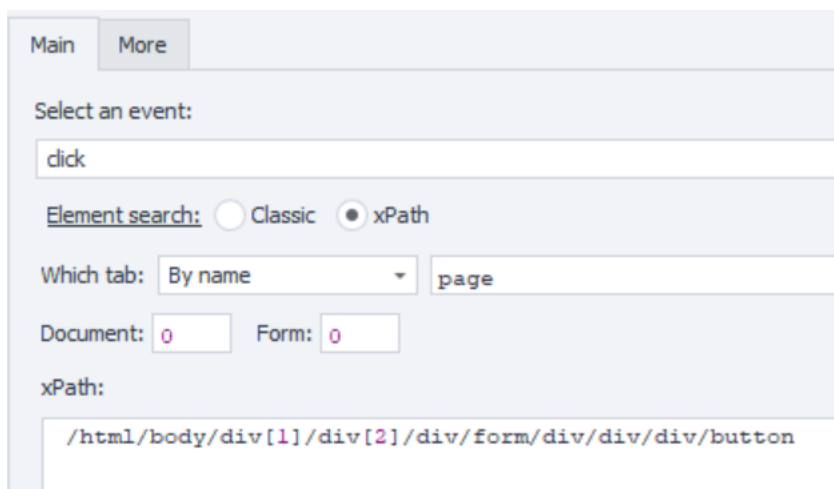
The screenshot shows the ZennoPoster application window. At the top, there are two tabs: 'Main' and 'More'. The 'More' tab is selected. Below the tabs, there is a section titled 'Select an event:' with a dropdown menu showing 'click'. Underneath, there is a section for 'Element search:' with two radio buttons: 'Classic' and 'XPath'. The 'XPath' radio button is selected. Below this, there is a section for 'Which tab:' with a dropdown menu showing 'By name' and a text input field containing 'page'. Further down, there are two input fields for 'Document:' and 'Form:', both containing the number '0'. At the bottom, there is a section for 'XPath:' with a text input field containing the XPath string: '/html/body/div[1]/div[2]/div/form/div/div/div/button'.

Figure 24 - ZennoPoster XPath for captcha button

Again, following the XPath the application understands which button to press. It is better to use XPaths or CSS selectors for any button clicking or text extraction, because pages often change and using coordinates is not reliable.

Further, a page simulation is executed. This is a built-in feature of ZennoPoster that simulates page browsing. It accurately tries to match a user’s mouse cursor and moves to big buttons and scrolls randomly. Unfortunately, yet this function cannot be modified with custom code, but it’s getting its job done. The simulation usually lasts for fifteen to twenty seconds after which the status of the captcha is checked again. It can be still in progress or in case it was successfully solved it will save the status in the status variable.



Figure 23 - Geizhals.at
Friendly Captcha page before
solving the captcha



In this figure, the captcha is still in progress and cannot be marked as solved. This is because the “abschicken” button is disabled and greyed out. These comparisons’ functions must be inserted by the developer.

Figure 25 - Geizhals.at Friendly Captcha solving in progress

Finally, when the captcha is solved, the status is saved to the project’s variables. While the captcha was being solved a loop function was checking the status every few seconds. If the status of the captcha is marked as completed, the application will simulate another click on the “abschicken” button using xPaths and proceed to extract the content and modify the task in the database.



Figure 26 - Geizhals.at Friendly Captcha solved status

```

var collectionName = project.Variables["collection"].Value;
var dbConn = ZennoLab.OwnCode.MongoConnection.getConnection(project);
var collection1 = dbConn.GetCollection<BsonDocument> (collectionName);
var filter = Builders<BsonDocument>.Filter.Eq("_id", new ObjectId(project.Variables["taskId"].Value));
var update = Builders<BsonDocument>.Update
    .Set("status", "FINISHED")
    .Set("failLeft", project.Variables["failure"].Value)
    // .Set("used", false)
    .Set("date", DateTime.UtcNow)
    .Set("crawledContent", instance.ActiveTab.GetSourceText("UTF-8"))
    // .Set("crawledContent", Convert.FromBase64String(instance.ActiveTab.DomText))
    // .Set("crawledContent", Encoding.GetEncoding(1251).GetBytes(instance.ActiveTab.DomText))
    .Set("userAgent", project.Profile.UserAgent);
collection1.UpdateOne(filter, update);

```

Figure 27 - ZennoPoster C# code to save the crawled content into the database

The task is marked as finished and the crawled content is extracted from the current tab's document.

```

1 {
2   "_id" : ObjectId("6257dcd4b0245d48f84de66c"),
3   "proxy" : "168.119.5.57:25217",
4   "request" : "https://geizhals.at/?fs=123456789&hloc=at",
5   "status" : "FINISHED",
6   "marketplace" : "geizhals",
7   "creationTime" : ISODate("2022-09-09T09:32:00.150+0000"),
8   "params" : {
9     "proxy" : "168.119.5.57:25217",
10    "pages" : "1",
11    "method" : "simulation",
12    "pageurl" : "https://geizhals.at/",
13    "key" : "5af9e8f7089cb2cd4df6b6f3d81b150d"
14  },
15   "_class" : "de.pi.capmonsterbalancer.models.CrawlingTask",
16   "date" : ISODate("2022-05-16T10:11:02.960+0000"),
17   "crawledContent" : "<!DOCTYPE html><html lang=\"de\"><head style=\"\">\n<script type=\"text/javascript\">
18   "failLeft" : "",
19   "userAgent" : "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
20 }

```

Figure 28 - MongoDB solved task object

The ZennoPoster whole project consists of the following:

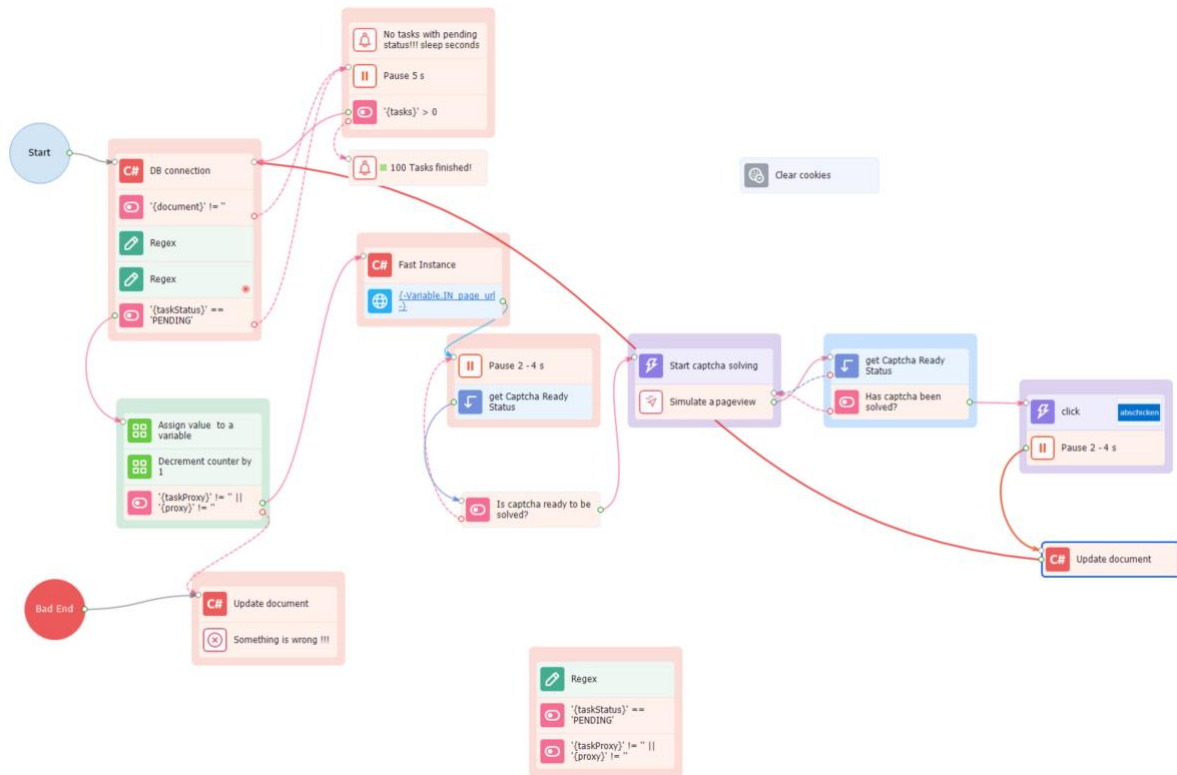


Figure 29 - ZennoPoster project overview

Above is the graphical user interface for the project's modules. C# modules represent custom code, green blocks are regex and comparison functions, blue and purple modules are requests and human simulation functions. Arrows represent a module's outcome. If it originates from a green point, it means that the previous module has succeeded and returned an expected value. If it originates from a red dot, it means that the outcome of the previous module was negative, and result was not as expected.

A mini project is needed to perform the never-ending loop for task fetching.

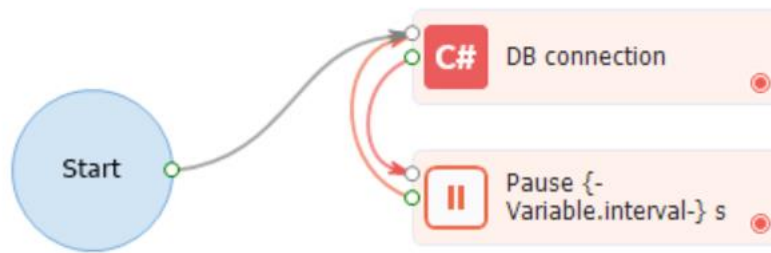


Figure 30 - ZennoPoster Task Fetching Project

This simple project checks for newly created tasks in the database with a set interval. The database connectivity is again retrieved from the shared code directives.

Finally, the projects can be deployed in the cloud and scheduled accordingly:

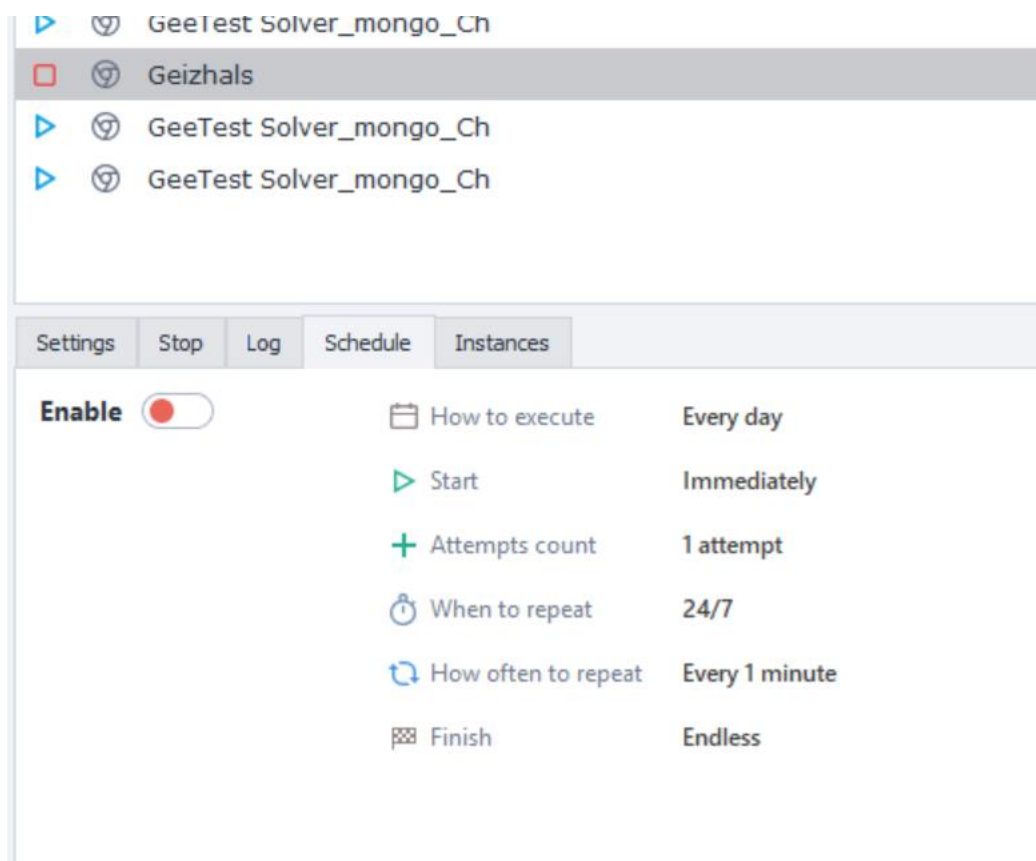


Figure 31 - ZennoPoster project scheduling settings

6.2 Ranking action

Back to our main application. After the ZennoPoster application has successfully solved the captcha, it is able to retrieve the content through the database. Using that content, it transforms it to an HTML document and proceeds to call the parser service.

The parsers make use of JSoup library to pick certain fields and get the correct data. They start by validating a page:

```
static boolean isRankingPage(Document document) {  
    return !document.select(OFFERS_CSS_SELECTOR).isEmpty()  
}
```

Figure 32 - ranking validation

If validation is passed, pagination is checked. In the context of crawlers' pagination means verifying all the available pages of a product page. There are websites like Google Shopping or Toppreise.ch that can have hundreds of offers on popular products which they divide into multiple pages. Pagination functions are responsible for identifying all such pages and adding them to the directory of crawled content.

```
static void doPaginationCrawl(CrawlingActionContext crawlingActionContext) {  
    int pageCount = getPagesCount(crawlingActionContext)  
    if (pageCount > 1) {  
        String paginationUrl = crawlingActionContext.getCrawlingDriver().getCurrentUrl() + NEXT_PAGE_QUERY  
        for (int i = 2; i <= pageCount; i++) {  
            CommonUtils.randomSleep(5000, 1000)  
            crawlingActionContext.getCrawlingDriver().get(paginationUrl + i)  
            crawlingActionContext.getCrawlingDriver().collectCrawledContent()  
        }  
    } else if (crawlingActionContext.getCrawlingDriver().hasElement(MORE_OFFERS_CSS_SELECTOR)) {  
        String moreOffersText =  
            crawlingActionContext.getCrawlingDriver().findElement(MORE_OFFERS_CSS_SELECTOR).getText()  
        if (StringUtils.containsIgnoreCase(moreOffersText, MORE_OFFERS_TEXT)) {  
            CommonUtils.randomSleep(5000, 1000)  
            crawlingActionContext.getCrawlingDriver().get(crawlingActionContext.getCrawlingDriver().getCurrentUrl() + "?  
tableonly=1&pg=2")  
            crawlingActionContext.getCrawlingDriver().collectCrawledContent()  
        }  
    }  
}
```

Figure 33 - Pagination function

After all the pages have been collected and validated it is time to pick apart every single html element that contains desired data. Let us start with the product's metadata.

```
private void setProductInfo() {
    BaseProductDetails productDetails = new BaseProductDetails()
    Map<String, Object> details = new HashMap<>()

    result.setProductUrl(GeizhalsUtils.parseProductUrlFromRanking(document))
    result.setProductName(GeizhalsUtils.parseProductNameFromRanking(document))
    result.setProductNr(getProductNrFromUrl(result.getProductUrl()))
    result.setImageUrl(GeizhalsUtils.parseProductImageUrlFromRanking(document))
    result.setCategory(document.select(CATEGORY_CSS_SELECTOR)?.first()?.parent()?.select("a)?.text()?.trim())
    result.setProductDescription(document.select(DESCRIPTION_CSS_SELECTOR)?.text())

    details.put("eans", extractEanList())
    productDetails.setDetails(details)

    result.setProductDetails(Collections.singletonList(productDetails))
}
```

Figure 34 - product metadata setting function

Information like a product's URL, name, image URL, category and description are very useful for later analytics. Most of the extracting functions make use of CSS selectors to get the right element:

```
static String parseProductNameFromRanking(Document document) {
    String productName = document.select(PRODUCT_NAME_RANKING_CSS_SELECTOR).first()?.text()?.trim()
    if (StringUtils.isBlank(productName)){
        productName = document.select(PRODUCT_NAME_SEARCH_CSS_SELECTOR).first()?.text()?.trim()
    }
    return productName
}
```

Figure 35 - Product name parser function

After product information has been extracted it is time to loop through its offers and save each one of them. Usually offers contain much more information than product's metadata and are the most sought after by customers. As a company you are more interested to see what other vendors are selling you product and what prices or sales are they having now. It allows you to have a competitive advantage and big overview on the competition.

```

private List<BaseOffer> extractOffers() {
    List<BaseOffer> offers = new ArrayList<>()
    String newDomain = document.select(GeizhalsUtils.DOMAIN_CSS_SELECTOR).attr("content")
    domain = !newDomain ? domain : newDomain
    Elements trElements = document.select(GeizhalsUtils.OFFERS_CSS_SELECTOR)
    for (Element trElement : trElements) {
        parseOffer(trElement)

        offers.add(offer)
    }
    return offers
}

```

Figure 36 - offer extraction function

After extracting the product's metadata and offer's data, validation functions are in place. To keep the database as homogenous as possible, it is vital to check for any errors during the parsing.

If everything checks out the product information and offers are saved in the database. The goal was met. Using a EAN as an input, a product was found, and data was successfully extracted and saved.

```

{
  "_id" : ObjectId("60b65b7f2193005f40dcc024"),
  "crawl_id" : ObjectId("627db14e04cf2efa80f73c8c"),
  "modificationDate" : ISODate("2022-05-14T01:17:07.487+0000"),
  "key" : "geizhals.de_c3b4f59ec7baf0ec06c37e209d646658",
  "ean" : "805289115700",
  "eanType" : "EAN",
  "name" : "Ray-Ban Sonnenbrillen RB3025 002/58 62 mm/14 mm",
  "url" : "https://geizhals.de/redir.cgi?h=edeloptics-at&loc=https%3A%2F%2Fwww.edel-optics.at%2FAVIATOR-LARGE-METAL-(RB302",
  "merchantId" : ObjectId("60b65b7f2193005f40dcc01b"),
  "merchantName" : "edel-optics.at",
  "marketplaceName" : "geizhals.de",
  "sellerName" : "edel-optics.at",
  "platformName" : "WEBSHOP",
  "listing_ids" : [
    ObjectId("60b65b7f2193005f40dcc017")
  ],
  "currentPrice" : {
    "creationDate" : ISODate("2022-05-14T01:17:07.487+0000"),
    "modificationDate" : ISODate("2022-05-14T01:17:07.487+0000"),
    "shippingPrices" : [
    ]
  },
  "shopRatingCount" : NumberInt(0),
  "expired" : true,
  "buybox" : false,
  "prime" : false,
  "cheapestOffer" : false,
}

```

Figure 37 - MongoDB saved offer object

```

{
  "creationDate" : ISODate("2022-04-30T01:16:59.405+0000"),
  "value" : 108.64,
  "currency" : "EUR",
  "rank" : NumberInt(1),
  "availability" : NumberInt(1),
  "deliveryDays" : NumberInt(-1),
  "deliveryDaysMax" : NumberInt(-1),
  "shippingPrices" : [
    {
      "paymentTypes" : [
        "vorkasse",
        "kreditkarte",
        "paypal",
        "lastschrift"
      ],
      "shippingCost" : 0.0,
      "productPrice" : 108.64,
      "totalPrice" : 108.64,
      "currency" : "EUR"
    }
  ],
},

```

Figure 38 - MongoDB offer price object

7 Conclusion

Project's expectations were met and successfully executed. Using a list of articles' numbers, the application is now able to search for them on the geizhals.at marketplace and bypass the captcha. The task was comprised of not only code writing, but also database designing and ZennoPoster development.

Geizhals is one of the most guarded websites in Europe's marketplace. Companies are spending tons of money of third-party services to protect their data. This project not only accomplished its mission but also demonstrates how vulnerable are modern ecommerce websites and how valuable their data can be. So far there has not been a single marketplace that succeeded in protecting its data. This shows the lengths to which we progress to defeat the big player's systems and extract the public data.

Someone quoted that data is today's gold and I think it is true. Who ever has the data – has the power to change things and influence others. Money rules the world and staying on top of knowledge can make the difference between high sales and bankruptcy. The desire to know more and be able to access data is growing day by day. People, small businesses, and large organizations are always trying to be on the edge of everything. What is my competitor doing? How can I automate my tasks? Which KPIs can help me improve my sales? These are just a few questions that I am sure every for-profit company's director is asking himself.

These pieces of software may not be the best options available, but it showcases what software engineering means. Just like an architect gets the materials, then designs, and constructs the bridge, a software engineer designs and crafts system that work simultaneously together to achieve a common goal.

Personally, I feel accomplished with the project. I learned a lot and implemented cutting edge technology, scaled it up and made it available for the Priceintelligence.

8 Bibliography

- Priceintelligence GmbH. (n.d.). *Priceintelligence*. Retrieved from Priceintelligence: <https://priceintelligence.net/en/about-us/#company>
- W3 Schools. (n.d.). *CSS Selector Reference*. Retrieved from W3 Schools: https://www.w3schools.com/cssref/css_selectors.asp
- W3 Schools. (n.d.). *CSS Selector Reference*. Retrieved from W3 Schools: https://www.w3schools.com/cssref/css_selectors.asp
- Geizhals. (2022, July 4). *Das Unternehmen*. Retrieved from Geizhals Pricevergleich: <https://unternehmen.geizhals.at/>
- ots.at. (2022, July 4). *Geizhals.at*. Retrieved from ots.at: <https://www.ots.at/pressemappe/2033/geizhals-at-preisvergleich-internet-services-ag>
- Built With. (n.d.). *CAPTCHA Usage Distribution in the Top 1 Million Sites*. Retrieved from Built With: <https://trends.builtwith.com/widgets/captcha>
- ProWebScraper. (2022, June 9). *Top 10 Captcha Solving Services Compared*. Retrieved from ProWebScraper: <https://prowebscraper.com/blog/top-10-captcha-solving-services-compared/>
- Ahn, L. v. (2017, October 27). *The reCaptcha Project*. Retrieved from Carnegie Mellon University: <https://web.archive.org/web/20171027203659/https://www.cylab.cmu.edu/partners/success-stories/recaptcha.html>
- Lalani, S. (2021, November 12). *10 Best Java Web Crawling Tools And Libraries In 2021*. Retrieved from XPERTI: <https://xperti.io/blogs/java-web-crawling-and-scraping-libraries/>
- GO FAIR. (n.d.). *FAIR principles - GO FAIR*. Retrieved from GO FAIR: <https://www.go-fair.org/fair-principles/>
- Friendly Captcha. (n.d.). *Friendly Captcha*. Retrieved from Friendly Captcha: <https://friendlycaptcha.com/>