Wrocław University of Science and Technology
Faculty of Electronics, Photonics and Microsystems

FIELD OF STUDY: Electronic and Computer Engineering (ECE)

BACHELOR THESIS

TITLE OF THESIS:
A web application that helps in the organization and analysis of offers from e-commerce stores.

AUTHOR:
Konrad Błaszczak

SUPERVISOR:
dr inż. Wojciech Domski, K29W12ND02

WROCŁAW 2021

Contents

Chapter 1

Introduction

With a development of internet, digital payment, transport, logistics, security, processing and storage of data, number of e-stores have highly increased in last years [4]. Nowadays, almost every company that works in retail has its own selling platform in the internet, not to mention big businesses such as Amazon, eBay or Aliexpress that have major share in e-commerce market [4]. As a client it is impossible to track desired product on all platforms or to analyze its price in longer period of time e.g. to predict when another discount will set or to check if store uses shady practice of rising a price of product then to discount it to normal price and label it as promotion.

Solution to this problem is to gather data periodically from this websites and save it in one place then analyze to achieve desired results. There is a technique in programming called web scraping. Scraping a website means to download a website structure and extract valuable data. Once fetched content of a page can be parsed (analysis of string of symbols), reformatted, searched and sent to spreadsheet or database. Web scraping usually takes information from website code and reuse that data in a different purpose for example like previously mentioned problem with price of products. Scraping can be used to periodically gather information from desired shop and save that data locally within database. Web pages are built using text-based languages such as HTML, JavaScript, and frequently contain an useful and valuable data in text form. Unfortunately, almost all websites are designed for human end-users and not for machines. As a result there are many specialized tools and software used to prevent bots from scraping a website. To prevent detection web scrapes use computer vision or natural language processing to simulate human browsing to enable content gathering from webpages. Most sites have minimal security in the form of Cloudware services or/and captcha to prevent bots from their servers. This is important in web scraping because the program sends a request to the site, and that means the more data is needed, the more requests will be sent to the site. In conclusion if a bot loads site with too many requests, previously mentioned security measures will be triggered and will prevent any further scraping, and in worst case scenario administrator of the site can ban bot by its IP address and permanently prevent any future scraping of a given website.

Web scraping involves a various problems such as I/O mechanism, multithreading, communication, task scheduling, duplication and many more. Coding language, framework and used packages will have a significant impact on how effective scraper is. To choose a proper language, few things were considered to be look for: flexibility, ability to feed database, scalability, maintainability. Python is mostly know to be best language for web scraping application. It can handle many web scraping related processes smoothly. Requests library is a standard for making a HTTP requests in Python, with that request can be sent for website and as a result it whole code is obtained in text form for further modification. Beautiful Soup is one of the most recognizable frameworks for web scraping used in Python language. It was designed for a fast and highly efficient web scraper. Notable features are pythonic idioms for navigation, searching and modifying a parse tree. It works with popular parsers such as lxml or html5lib which means there is possibility to try different approaches for parsing website.

This project needs to collect data from internet and manage database to storage collected data in some form. In Python data can be storage in list, dictionary, set or tuples. List is used to store multiple items in one variable, list items are ordered, changeable, and allow duplicate values. Dictionary is used to store data values in key:value pairs and it is a collection which is ordered, changeable and does not allow duplicates. Set is used to store multiple items in a single variable and it is a collection which is unordered, unchangeable, and unindexed. Tuple is used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable. For collected data in this project the most suitable form of storing data was dictionary. Due to that fact the best database will be Elasticsearch which saves data as JSON documents. Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows to store, search, and analyze big volumes of data quickly and in near real time. Elasticsearch provides system based on top of Apache Lucene written in Java for indexing and automatic type guessing which is suitable for this kind of data. Best advantages of Elasticsearch are:

- scalability,
- fast performance,
- multilingual searching,
- documents in JSON format,
- auto-completion and instance search.

Elasticsearch has also one very important feature that is key for this project. It has dedicated Python client REST API, which means Python script can communicate directly with Elasticsearch without any third party software.

Web app is software application that runs on a web server. Web applications are accessed by the user through a web browser with an active network connection which means controlling code or getting results from any device. Beacuse project was written in Python two framework were considered: Django and Flask. Django is better suited for this project due to being full-stack web framework. Django's biggest advantage is built-in basic functionalities that in Flask must to be written from scratch or to be installed from external source.

1.1 Thesis

The main goal of this project is to collect monthly worth data from e-commerce store, observe changes in the products and analyze data to indicate if product is worth buying.

Chapter 2

Data

2.1 Acquiring data from websites

Today, the main component of almost every website is the HyperText Markup Language, or HTML for short. It is used to display documents in a web browser and is very often used with other technologies such as Cascading Style Sheets (CSS) and scripting languages like Java Script or PHP. When a browser loads a web page it creates an document object model based on HTML. See Fig.1 for more details.

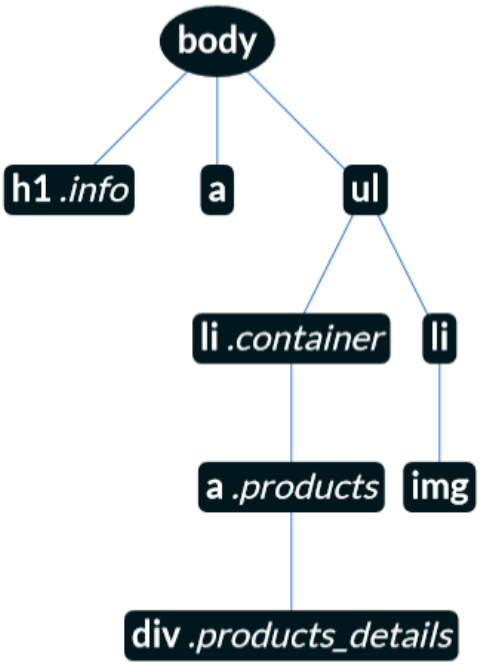

Figure 1 Document Object Model

This structure makes it possible to access individual page elements. An element can be found in several ways: by a class identifier, by a tag name, by a class name, by CSS indicators or by a group of HTML elements.
To get the document object model, request to the page server is required by using either HTTP (Hypertext Transfer Protocol) or HTTPS (Hypertext Transfer Protocol Secure). Both protocols belong to the application layer in the ISO/OSI model. The difference between the two is that the HTTPS protocol is encrypted with a TLS (transport layer security) or SSL (secure socket layer) technology.

The HTTP protocol has several types of requests that can be send to the server:

- GET is used to retrieve the whole resource indicated by the URL,
- POST is used to receive data from a client to a server,
- PUT is used to receive data from the client to the server,
- PATCH updates page resource data,
- HEAD is a request that retrieves information about a resource,
- DELETE is a request to delete the selected resource,
- TRACE is used to analyze the communication channel,
- OPTIONS informs about available options and requirements of the, communication channel,
- CONNECT is a request intended for proxy servers.

In Python, the most popular package for sending HTTP requests is the Requests package created in 2011 by Kenneth Reitz and published on the Apache 2.0 license [5]. The Requests library uses the same key verbs as the HTTP protocol. This means that the method names used by the HTTP protocol and the Requests package are the same. In this project, the Requests package was used to retrieve through the GET method a document object model containing a list of products from e-commerce store.

2.2 Collecting and data parsing of the document object model

The goal of the project was to collect as much data as possible from the online store. The first obstacle was to collect all URLs containing the list of products, because almost all online stores operate on categories and subcategories. It happens that even subcategories have their own subcategories which prolongs the process of redirecting to the product page. The solution to this problem was to write a script that checked whether a given URL contained a list of products. The program ran on 2 lists and 10 threads running in a while loop. First list collected all URLs of categories and subcategories that are available on the site, while the second list collected all URLs, which had a class with a special identifier, which contained a list with products and number of pages. The condition to be added to a particular list was the existence of previously mentioned class with products. If the condition was true the URL was put in the 2nd list and removed from the 1st list, if the condition was false it meant that other subcategories were available, so the script was collecting all URLs of subcategories, adding them to the 1st list and removing the checked URL. The loop ended when 1 list was cleared of all URLs. It then saved all the URLs from the 2nd list including the maximum page count information to a CSV file. In this way it was able to significantly reduce the data collection time.

The second problem was the number of requests sent to the server. All the URLs containing products after the scraping were in number on 1330. Some websites allow to increase the number of displayed products per page, by passing e.g. limit attribute in the URL.

This procedure meant to decrease in the number of pages, which translates into the number of HTTP requests sent to the server during daily data collecting. Online stores very rarely change their layout what means that this script does not have to be run each time before actual data collecting from products.

The data that was collected for each product included: name, description, price, currency, position on the page, link to the product, rating and voting. Like the previously mentioned script, the main data collector also operated on 10 threads. The code sent 10 requests to the page, if the status was 200 it meant that the page was reachable and its data could be collected. Otherwise, the thread dropped the address and waited for a new one. The interval between subsequent requests to the server was 2 seconds in order not to overload the page server. The main data collector used BeauifulSoup and the lxml parser to search for the previously mentioned attributes. Each attribute required minor editing adjustments such as deleting unicode characters, extra spaces, or characters marks that start a new line. After collecting all data from a given URL, the product lists are saved to CSV files.

2.3 Multithreading in speeding up the data collection process

Multithreading is a feature of operating systems that allows several tasks to be performed within a single process. After the introduction of multithreading, the data collection time was significantly reduced. When the script collecting data was running on a single thread, the working time was 6 hours and 34 minutes, while on ten threads the program finished working in 40 minutes working with the same dataset. The advantages of a multithreaded program over a multiprocess program were:

- many threads in the same process communicate with each other faster and more freely than processes with each other. This is because all threads in a process share the same virtual address space, meaning they have access to the same variables or objects and use the same system resources,
- threads need less memory to operate than separate processes,
- the work of threads can be divided into 3 phases: the invocation of the thread, the execution of the task and the return of the result, threads also have a command counter that tracks where the thread is running,
- a thread can be put to sleep or interrupted without affecting concurrent threads running at the same time.

An important factor in implementing a multi-threaded system was to sort all URLs by the maximum number of pages. When URLs were not sorted, a situation occurred where 9/10 threads finished their work in 10 seconds, while the last thread still had time of work left. This caused unnecessary downtime in the application. Thanks to sorting addresses retrieved by the code had similar number of pages in each run, which eliminated the downtime.

Chapter 3

Database

Elasticsearch is an open source server search engine based on the Lucene library that can be interacted with through a RESTful web interface API. Elasticsearch stores documents based on JSON (JavaScript Object Notation) technology. In this project, the Elasticsearch was used to store and quickly access data collected from an e-commerce store. Documents are the smallest pieces of data, while not considering fields in them. Documents are stored in shards. There are 2 types of shards in Elasticserach. Primary shards that contains data and replica shards that are a copies of primary shards. A Sharding system is used to evenly distribute data and to spread a load. Shards are collected into indexes and allocated in nodes. Primary and replica shards cannot be stored in one node, because replica shards substitute primary ones in case of node failure. Group of nodes with the same *cluster.name* attribute is called a cluster, as nodes join or leave the cluster recognize itself to automatically redistribute data across all available nodes without user intervention. It is important to keep cluster status in good condition. There are 3 statuses that indicated cluster health. Green indicates that everything is correctly allocated and ready for searching. Yellow indicates that one or more replica shards are not allocated. In this status there is not loss of data while searching, but problem should be resolved as soon as possible. Red status indicates that one or more primary shards (and all of its replicas) are not properly allocated, in this status there is loss of data while searching, and indexing a documents in this shard will return error.

3.1 Data indexing

To index data Elasticsearch uses structure called an inverted index. This structure allows to perform fast full-text searches [3]. Inverted index contains a list of all unique words that appear in any document and for each word it contains a list of documents in which it appears. For proper work inverted index needs to know about all documents in collection. Inverted index does not need only to contain this two lists, it can also contain much more e.g. number of times each word appeared in a document, the order of terms or average length of all documents. Inverted index stored on a hard drive is constant which means it cannot be changed, but this immutability has benefits which cause that inverted index to be of high performance during read operations.

To make Inverted index updateable a concept called pre-segment search from Lucene was used. In Lucene a segment is an inverted index, and word index is a collection of segments plus commit point – a file that list all segments [3]. Pre-segment search is cre-

ating a new segment from in-memory buffer with overwritten commit point. There is no possibility to update or delete segments, only possibility to "remove" or "update" segments is to update a commit point and as a result old segments will not appear in query results.

To send the gathered data to Elasticsearch a Python program was written. It used official Elasticsearch client for Python. All data gathered by script from chapter 2 were in CSV files. Program had two inputs from user. It were path to data and date from which day that data came from. By setting this two variables program was deleting empty CSV files and merging other documents that contained data into one variable. Then program was creating an Elasticsearch index with previously annotated name, settings and mappings.

For each day one index was created named after name of an e-commerce store from which data was collected and timestamp to indicate from which day the data came from. Each index had 1 shard and 1 replica. This solution was made to easier troubleshoot, update data and have fewer restrictions on data mapping.

Each document in Elasticsearch contains 3 metadata: *_index,* *_type* and *_id*. Index indicates from where document comes from. Type, as the object class, is use to create a separate collection inside the same index but this solution is deprecated and will be no longer supported in newer versions of Elasticsearch. All documents in this project had *_doc* type. ID is an unique identifier for each document. This metadata also can be set by the user but to avoid repetition autogenerated ID was used instead.

3.2 Mapping and analyzers

For each index, it was important to map the fields properly and use correct text analyzers for the text type fields. Elasticsearch does not have official support for the Polish language analyser, the producer recommends using The Stempel Analysis plugin, which detects plurals of words e.g. searching for the word "telewizory" will find documents that contain that word in them, both singular and plural form without the plugin results would only contain the plural form. The plugin had also a stop list, which is a list of words rejected by the search engine in order to reduce the size of the set. Such words that are available in this plugin are for example conjunctions, academic titles or personal pronouns.

Field name	Type	Analyzer
name	text	polish
deatails	text	polish
price	float	X
currency	keyword	X
popularity	integer	X
rating	float	X
vote	integer	X
link	keyword	X
date	date	X

Table 1. Mapping used in all indexes.

The difference between keyword and text types can be seen when saving them in inverted index. Fields of type text are analyzed before saving. By default it is a standard analyzer that splits words according to the rules defined by the Unicode Consortium, removes punctuation and reduces all text to lowercase in order to improve their "searchability". Fields of the keyword type are not subjected to such analysis.

Example of text analyzed by the plugin stempel. *Name* of product: Konsola SONY PlayStation 5 + 2 Kontrolery DualSense Czarne + FIFA 22 + PlayStation Plus 365 dni.

token	start_offset	end_offset	type	position
konsola	0	7	<ALPHANUM>	0
son	8	12	<ALPHANUM>	1
playstati	13	24	<ALPHANUM>	2
5	25	26	<NUM>	3
2	29	30	<NUM>	4
kontroler	31	41	<ALPHANUM>	5
dualsens	42	51	<ALPHANUM>	6
czarny	52	58	<ALPHANUM>	7
fif	61	65	<ALPHANUM>	8
22	66	68	<NUM>	9
playstati	71	82	<ALPHANUM>	10
plus	83	87	<ALPHANUM>	11
365	88	91	<NUM>	12
dni	92	95	<ALPHANUM>	13

Table 2. All tokens after analysis.

After analysis, each such token is send to the list in the inverted index. An example decomposition can be found in Table 3. Inverted index has 3 columns: *term*, *frequency* and *document ID*. Tokens are assigned to *term*. *Frequency* is used in term frequency indicator that helps calculate relevance which in turn is the key factor in the data mining

[1]. Lastly the *document ID* is used in estimating in which document the term appears. Inverted index is created for each field.

Term	Frequency	Document ID
2	1	1
22	1	1
365	1	1
5	1	1
czarny	1	1
dni	1	1
dualsens	1	1
fif	1	1
konsola	1	1
kontroler	1	1
playstati	2	1
plus	1	1
son	1	1

Table 3. Inverted index

3.3 Data searching

There are two most important factors considered in searching data with Elasticsearch. First one is relevance which is to estimate how relevant are results to a given query. It is calculated with Elasticsearch similarity algorithm which is ratio of Term Frequency and Inverse Document Frequency (TF/IDF) [3]. Term Frequency parameter is higher for documents, in which a term appears multiple times. Inverse Document Frequency terms are less relevant if there are many of them in an index. This parameter is highest for terms that are most unique in index. Another factor taken in estimating a relevance is field-length norm. Terms in short fields carries more weight to algorithm than in long fields. For performance reasons Elasticsearch does not calculate TF/IDF for each document but for each shard. If data is not evenly distributed then same words will have different score in each shard. Problem fades when more data appear in the index but to be sure that data is evenly distributed in this project each index contained only one primary shard and one replica shard. In Elasticsearch relevance is estimate by field *_score* in metadata.

Second factor is analysis. It is process to decompose text into separated, normalized tokens by chosen analyzer in order to create an inverted index. All queries perform relevance calculation but not all perform analysis. Textual queries can be separated into two groups term-based and full-text. Term based queries are the ones that not perform analysis part. This type is looking for exact term in inverted index.

Full text queries like *match* that was used in the project can understand a mapping of the field. If query field is a keyword, the query string is treated as an exact value. If query field is a text field then query is passed through same analyzer as a field to produce a list of tokens that will be used in the search.

Match and *bool* queries were mostly used in this project to search trough data in Elasticsearch. *Match* query checks a field type if it is a text field or a keyword field. If It is a text field then analysis is performed for a query string. To find matching documents and score them. For searching multiple fields at once *bool* query was used. *Bool* query accepts multiple queries clauses under 3 parameters: *must*, *must_not* and *should*. *Must* and *must_not* parameters are simple bool filters. Should parameter tells that indicated terms are not required but if a document contains them it becomes more relevant in the search. To calculate relevance score *bool* query adds all scores from matched *must* and *should* clauses and then divide it by total number of clauses. Clause *must_not* is not consider in estimating on of a relevance score its only purpose is to exclude irrelevant documents.

Chapter 4

Collected Data

The main contribution of the project was to have a history of prices for a given set of products, in addition to collecting information about the display position on the page, user reviews (star system with grades from 0 to 5) and the number of votes that formed such reviews. Data collected in this project were from one e-commerce shop from 1.11.2021 to 24.12.2021. Data were collected within daily routine. The collected set is missing data from days 7.11.2021, 10.11.2021 and 23.12.2021 due to reasons beyond the author's control. Data for each day was collected as CSV files. All data were send to Elasticsearch and properly indexed with Python script to enable fast searching through it.

4.1 Data analysis and visualization

When the data was collected now it was possible to create visualization (Fig. 2) for each product to track it changes in time and analysis e-commerce shop behavior. colorcyan To visualize product history a term query with link was created to search for product in all indexes. . Price history can indicate whether the promotion is credible e.g price was not raised a few days before the sale. With price data gathered from longer period of time there would be possibility to check how often product is set on the sale. Popularity history can indicate if customers are interested in product on a daily basis and not only in time of the sale. Combining price and popularity data client can estimate if a product was set on the sale because it was not that popular with original price. By looking on rating and voting history client can estimate if rates were high due to voting by real people or by bots, or in a opposite situation client can check if product was under review bombing.

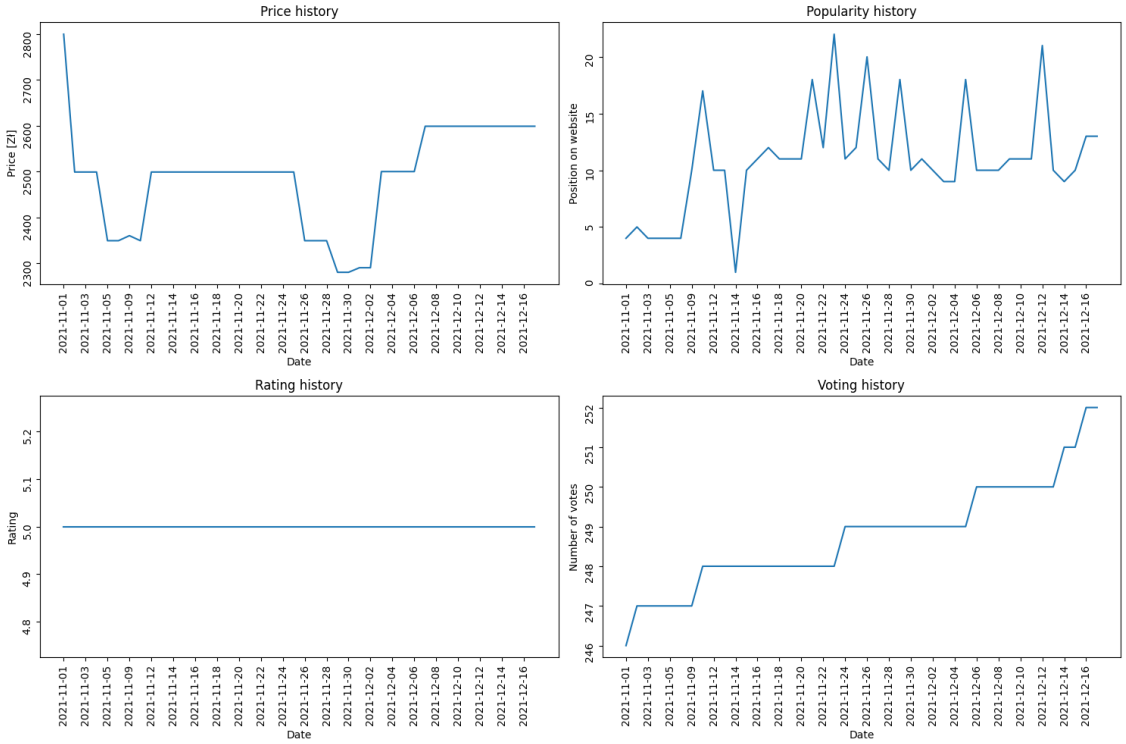

Figure 2. Charts displaying history of product

Data analysis is a process of data processing to derive useful information and conclusions from it. From the collected data where a sample was displayed in the Figure 2, analysis can be perform to recognize patterns and trends that can be useful during procurement decision. There is a possibility to seemingly unrelated data turn into one indicator and estimate a trend for the product. In this project 2 indicators were created. The purpose of the indicators was to asses whether changes over time will be positive or negative and have a standardized way in product comparison. First indicator was created as a sum of multiplied values,

$$f_1(x) = \frac{200 * 1}{x_1} + \frac{1,5 * 1}{x_2} + \frac{x_3}{5} + \frac{1,5 * x_4}{100} \tag{4.1}$$

where: x_1 is product price, x_2 is product popularity, x_3 is product rating, x_4 is vote rate for the product. Low price and high popularity was most influential in creating this indicator because these data are the most volatile. After data analysis, the coefficients were empirically selected. This indicator can be good for comparing two products with similar features such as product price but it cannot be used for comparison of products which have significant discrepancy of features. There are many flows in this indicator such as being favorable to products that has a small price tag. Popularity in this indicator has big influence only if position of product belongs to the first ten. For products that are displayed on further pages this variable will close to non-influence on the indicator. To device a better indicator that can be used for all types of products, data normalization must be introduced. Normalization is a procedure for pre-processing data to enable cross-comparison and further analysis. For this purpose a second indicator was introduced.

$$f_2(x) = -\frac{5 * (x_1(t+1) - x_1(t))}{x_1(t)} - \frac{0.4 * (x_2(t+1) - x_2(t))}{x_2(t)} + \frac{x_3(t)}{10} + \frac{1.5 * (x_4(t-1) - x_4(t))}{x_4(t)} \tag{4.2}$$

where variables are marked in the same way as in (4.1) equation. By calculating price, popularity and vote and change scaling them with a reasonable set of weights' the second

indicator can be used for comparison of different products. Changes such as price drop, higher position on page or additional votes might happen if trend is increasing. For first indicator trend is mostly depended on price changes. By having this two indicators and raw data visualization one can wait with decision of buying this product and wait for more favorable conditions. In the Figure 3 the first indicator was visualized. It can be seen the trend is decreasing, this situation is caused by the price drop. At the same time the second indicator (Fig. 4) is increasing. This means that the price is rising, but customer can expect promotion in the near future.

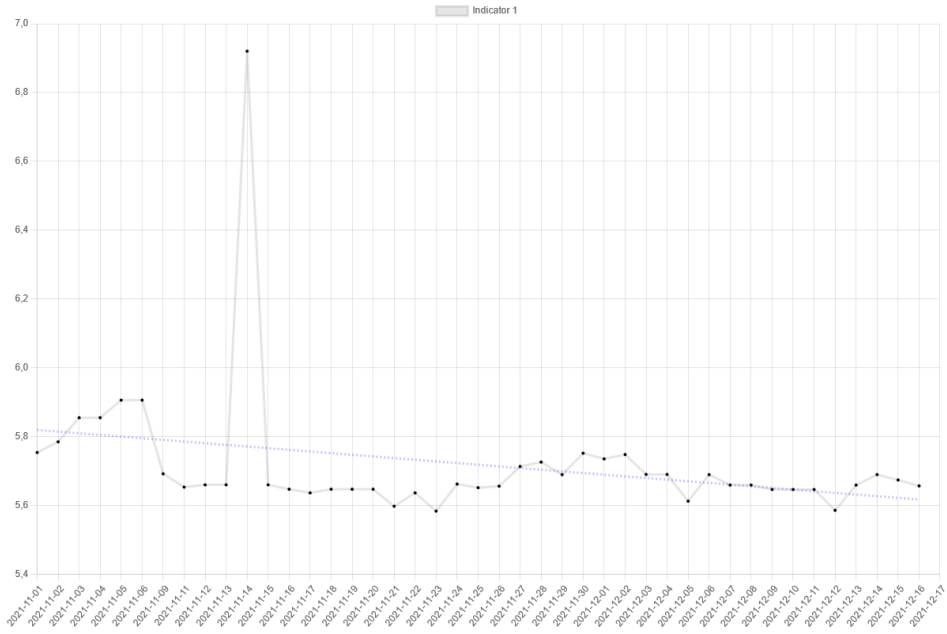

Figure 3. First Indicator

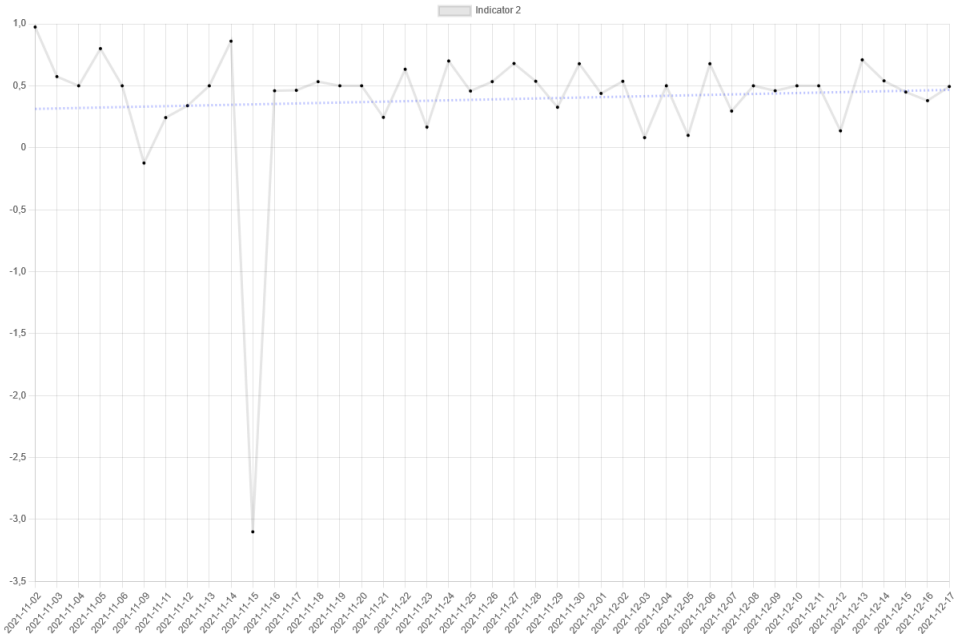

Figure 4. Second Indicator

Chapter 5

Web application

A web application, unlike a desktop application, does not require installation. Web application can be opened with an Internet browser to deliver interference for the user to present information. Biggest difference between web applications and websites is that websites are mostly used for information purposes and not to interact with an user. In this project it was important to create a simple user interface that can be accessible for everyone and from every type of devices. There are many frameworks that provide tools to create web applications with different programming languages. Ruby on rails is a framework that is using Ruby language. Angular JS and Express are using JavaScript language. There is a framework called ASP.NET that is using C# language. Due to the backend for the project was written in Python and communication with Elasticsearch was done through the official Python client, a Python-based framework was chosen. There are two major frameworks that were considered in this project. Django and Flask are two most popular framework that use Python language for creating web applications. For this project Django was used to create web interface for this project.

Django is high-level Python web framework that is used in creating web application. Today, it is one of the most popular web frameworks available, used by the largest websites in the world – Instagram, Pinterest, Bitbucket, DisqusIn [6]. In contrast to its competitor Flask it has already implemented functions that do not need to be written from scratch like admin interface, security and performance upgrades, database models, forms, URL routes, templates and user authentication. Django is based on *MVC* design pattern (*Model-View-Controller*). *Model* represent how data is organized in the database, *View* is what user see while visiting a website, Controller is for controlling a data flow in application including performing data analysis or more complex tasks on data retrieved from the *Model*. For Django model is named *MTV* (*Model-Template-View*) [6]. This model can be divided into two groups: client and server. Client is only able to see a *Template* and for server there are *Model* and *View*. Django uses requests and response via HTTP protocol to communicate between client and server sides. The View presents the model to the client as an HTTP response. In this project user was able to search data directly in Elasticsearch trough main page template. Two ways of searching data within Elasticsearch were implemented in the application. First one was with searching via link from e-commerce store to display its history and second one was to send customized query to retrieve all items from indexes which suited with query.

Data flow was presented in the Figure 5. User is directed to main page from where a form is displayed (Fig. 6) to be filled or left empty. After filling the form there are two

ways to interact with the web application. First path is to look for the product history. In this case only data that is sent to *View* is linked with the product. In the next step data is sent to function where it is assigned to previously prepared query and sent to Elasticsearch. Results are assigned to lists and returned to *View* where it is sent to the product history template to be displayed for the user. The second path is for products searching. It is similar to the first one but this query can be customized with more features. It needs to be send to "create query" function in order to construct query with the user input. The data flow is exactly the same as in the case of the first scenario. Query is sent to Elasticsearch and results are assigned to lists and then sent back to template. From both templates it is possible to return to main page and search for an another product.

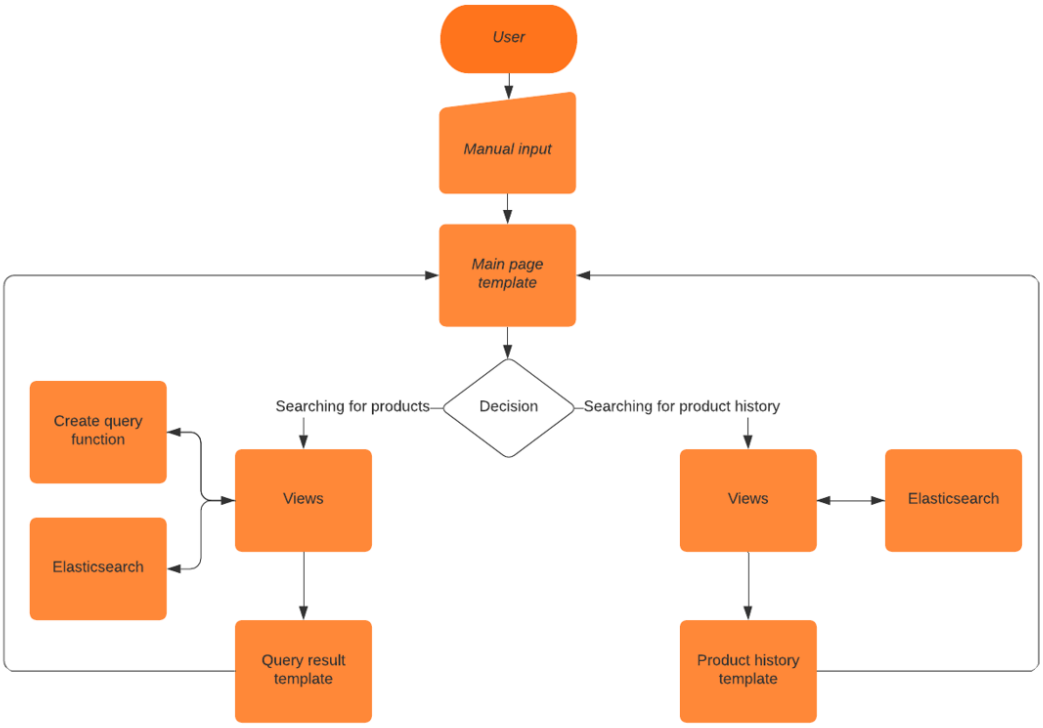

Figure 5. Web application flowchart

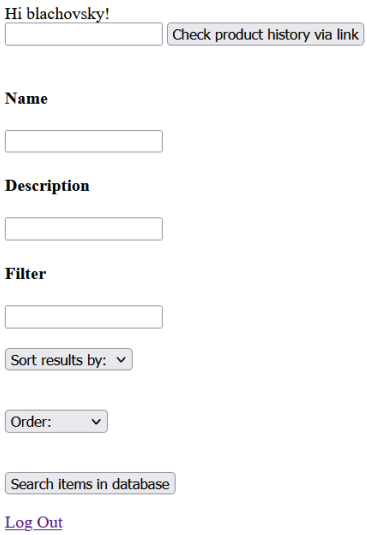

Figure 6. Main site

History of a product including its name, description and 4 charts that included how price, popularity, rating and votes changes over time. Each point can be highlighted to see detailed data (Fig. 7) To create interactive charts in templates Chart.js.

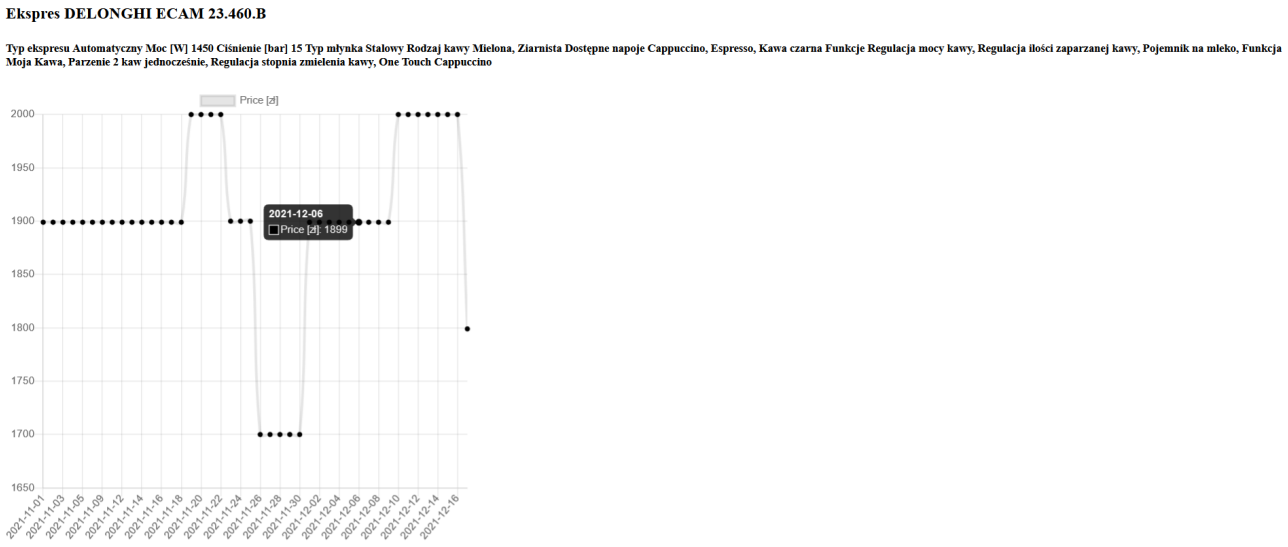

Figure 7. Price chart history for product

To seek a similar products in database a form from (Fig. 3) can be used. Filled form on main site sends the user input to create custom queries for Elasticsearch. Name and description were used for searching in respective fields name and details (Table 1) in Elasticseach indexes. Filter was used to remove unwanted results for example by searching a term "telewizory" also many accessories appeared for them such us remote controls or bases. Sorting was added to make searching more comfortable. Sorting was possible by price, popularity, rating and votes. Sorting order can also be configured. Example of displayed data:

```
Name: Ekspres KRUPS Evidence One EA895E
Description: Typ ekspresu: Automatyczny Moc [W]: 1450 Ciśnienie [bar]: 15 Typ młynka: Stalowy Rodzaj kawy: Ziarnista Dostępne napoje:
      Americano, Caffe Latte, Cappuccino, Doppio, Espresso, Herbata czarna, Herbata zielona, Latte Macchiato, Napar ziołowy, Ristretto,
      Spienione mleko Funkcje: Spienianie mleka, Regulacja mocy kawy, Regulacja ilości zaparzanej kawy, Wbudowany młynek, Tak, Dotykowy ekran,
       Parzenie 2 kaw jednocześnie, Regulacja stopnia zmielenia kawy, Regulacja temperatury kawy, One Touch Cappuccino
Price: 2599.0
Currency: zł
Popularity: 13
Rating: 5.0
Votes: 252
Date: 2021-12-17
Link: https://www.mediaexpert.pl/agd-male/ekspresy-i-kawa/ekspresy-cisnieniowe/ekspres-krups-ea895e-evidence-one


Name: Ekspres KRUPS Evidence EA8901
Description: Typ ekspresu: Automatyczny Moc [W]: 1450 Ciśnienie [bar]: 15 Typ młynka: Stalowy Rodzaj kawy: Ziarnista Dostępne napoje:
      Americano, Cappuccino, Doppio, Espresso, Gorąca woda, Herbata czarna, Herbata zielona, Latte Macchiato, Long Coffee, Napar ziołowy,
      Ristretto, Spienione mleko Funkcje: Spienianie mleka, Regulacja mocy kawy, Regulacja ilości zaparzanej kawy, Wbudowany młynek, Wskaźnik
      poziomu wody, Tak, Parzenie 2 kaw jednocześnie, Regulacja stopnia zmielenia kawy, Regulacja temperatury kawy, One Touch Cappuccino Kolor
      : Biały
Price: 1999.0
Currency: zł
Popularity: 17
Rating: 5.0
Votes: 306
Date: 2021-12-17
Link: https://www.mediaexpert.pl/agd-male/ekspresy-i-kawa/ekspresy-cisnieniowe/ekspres-krups-evidence-ea8901-bialy
```

Chapter 6

Conclusion

The purpose of this project was to design and develop a system that would allow the user to check the history and trends of a product in and accessible form which in turn would help in a process of making a buying decision. The project consisted of many elements. A backend script written in Python that was responsible for web scraping by separating link and data collecting, time of operation was significantly reduced. By adding many threads in data collecting this time was reduced even more. Database and search engine used in this project was Elasticsearch, which allowed to search over a million documents in a very short time. Due to fact that Elasticsearch is No-SQL database time was saved on creating high performance SQL database. User interface was created in Django framework to make it very simple and accessible for everyone. The data was collected from 1.11.2021 to 24.12.2021 with the exception of a few days resulting from problems beyond the author's control. The completed engineering project was dictated by the growing demand for sites such as fakefriday.org. With expansion of database on new shops and additional daily data whole system has prospects of development in many directions, where the most important one cohort analysis. Cohort is a term used in statistics and applied sciences, e.g. demography, to denote a set of objects, extracted from a set due to an event or process occurring simultaneously for the entire set for the purpose of analysis. This was a reason why Elasticsearch was used in this project. By full text search there is more freedom for extracting desired objects from set. Indicators presented in this project in future can be use for trend analysis which is a part of technical analysis to predict future trends more precise [2].

Bibliography

[1] C. C. Aggarwal. Data Mining: The Textbook, Springer International Publishing. 2015.

[2] J. D. Charles D. Kirkpatrick II. Technical Analysis The Complete Resource for Financial Market Technicians. 2016.

[3] Z. T. Clinton Gormley. Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine, O'Reilly Media. 2015.

[4] D. Coppola. E-commerce worldwide - statistics facts, Statista. 2021.

[5] R. Mitchell. Web Scraping with Python: Collecting More Data from the Modern Web, O'Reilly Media. 2018.

[6] W. S. Vincent. Django for Beginners: Build websites with Python and Django. 2018.

Appendix A

The paper is accompanied by a CD disk containing in individual folders:

- Digital copy of this work.
- Source code for data from Figure 4.
- Source code for web scraping.
- Source code for indexing documents.
- All data gathered from 1.11.2021-24.12.2021.
- Source code for Django web application.
- CSV file containing all URLs for e-commerce shop.