

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Aleksandr Lerko 202027

Exploring a Novel Approach for Aggregation of Company Profile Data

Bachelor's thesis

Supervisor: Pavel Tšikul
MSc

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleksandr Lerko 202027

Uue lähenemisviisi uurimine ettevõtte profiiliandmete korjeks

Bakalaureusetöö

Juhendaja: Pavel Tšikul
MSc

Tallinn 2024

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aleksandr Lerko

12.05.2024

Abstract

The problem of this thesis is exploring the possibility of using LLM for building an unstructured data website scraper as a part of the development of an application for aggregating company profile data registered in Estonia. The application is a prototype that is going to be tested using publicly available company profile data from the Estonian Business Register, and it has a potential for further development and expansion to use this application for processing companies registered in other EU countries.

The thesis aims to expand the Getpart OÜ database with a piece of information about other companies and to automate a process for aggregating companies into the Getpart application. It allows Getpart to greatly increase the company database and save many resources from time to money by automating this process.

During the development, the application was created using a microservices approach. It means that all the services created might be used independently from each other if necessary. The application contains 2 services and 1 gateway service, each for a specific task. They are working together with Gateway to get a final result, but, if necessary, each of them might be used separately for a specific task.

As a result, a working application prototype is capable of handling company data from the publicly available Business Register combining it with missing data from companies' websites using Open AI GPT-4 and aggregating it to a Getpart application.

This thesis is written in English and is 44 pages long, including 7 chapters, 9 figures and 3 tables.

Annotatsioon

Uue lähenemisviisi uurimine ettevõtte profiiliandmete korjeks

Käesoleva lõputöö probleemiks on uurida võimalust kasutada LLM-i struktureerimata andmeside veebilehe kaabitsa ehitamiseks osana Eestis registreeritud ettevõtte profiiliandmete korjaserakenduse väljatöötamisest. Rakendus on prototüüp, mida hakatakse testima Eesti Äriregistri avalikult kättesaadavate ettevõtteprofiilide andmete abil ning millel on potentsiaali seda skripti edasi arendada ja laiendada ka teistes Euroopa Liidu riikides registreeritud töötlemisettevõtetele.

Töö eesmärk on laiendada Getpart OÜ andmebaasi koos informatsiooniga teiste ettevõtete kohta ning automatiseerida ettevõtete korjaseprotsess Getparti rakendusse. See võimaldab Getpart oluliselt suurendada ettevõtte andmebaasi ja säästa palju ressursse aeg-ajalt raha, automatiseerides seda protsessi.

Arenduse käigus loodi rakendus, kasutades mikroteenuste lähenemist. See tähendab, et kõiki loodud teenuseid võidakse vajaduse korral kasutada üksteisest sõltumatult. Rakendus sisaldab 2 teenust ja 1 lüüsiteenus, iga konkreetse ülesande jaoks. Nad teevad koostööd Gateway'ga, et saada lõpptulemus, kuid vajadusel võib igäüks neist konkreetse ülesande jaoks eraldi kasutada.

Selle tulemusena on töötav rakenduse prototüüp võimeline käitlema ettevõtte andmeid avalikult kättesaadavast äriregistrist, ühendades need puuduvate andmetega ettevõtte veebisaidilt, kasutades Open AI ChatGPT-4 ja korjadesneed Getparti rakenduseks.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 44 leheküljel, 7 peatükki, 9 joonist, 3 tabelit.

List of abbreviations and terms

AI	Artificial intelligence
API	Application programming interface
B2B	Business to business service
e-Business Register	Estonian Company Registration Portal and the Visualised Business Register
CSV	Comma-Separated Values
DOM	Document Object Model
EMTAK	Estonian Classification of Economic Activities of the international harmonised NACE classification
EU	European Union
GPT	Generative Pre-Trained Transformer
GPU	Graphics processing unit
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
LLM	Large language model
ML	Machine learning
NLP	Natural language processing
REST	Representational State Transfer
RIK	Centre of Registers and Information Systems
SOAP	Simple Object Access Protocol
UI	User interface
URL	Uniform Resource Locator
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Table of contents

1 Introduction	11
2 Problem statement and purpose of the project	13
2.1 Problem statement	13
2.2 Initial conditions	14
2.3 Purpose	15
2.4 Methodology.....	15
3 Problem analysis.....	17
3.1 Related solutions.....	17
3.1.1 Open-Source solutions overview	17
3.1.2 Web services overview	18
3.2 AI language model choice	20
3.3 LLM based AI model comparison.....	20
3.3.1 Meta’s Llama overview	21
3.3.2 Google’s Gemini overview.....	22
3.3.3 OpenAI’s ChatGPT overview	24
3.4 Current implementation problems	24
3.5 A vision of the solution	26
4 Solution analysis.....	28
4.1 Solution requirements.....	28
4.1.1 Functional requirements	28
4.1.2 Non-functional requirements.....	29
4.2 Restrictions on the prototype	29
4.2.1 Incomplete data return	29
4.3 Technology choice.....	30
4.3.1 Open Data Service overview	32
4.3.2 AI Service overview	34
4.4 Various prompts comparison.....	35
4.4.1 Retrieving company description data	35
4.4.2 Comparing more sensitive data	36

4.4.3 Importance in accurately describing the prompt	37
5 Solution implementation	39
5.1 Application architecture	39
5.2 Aggregation service development	40
5.3 Open Data Service development	42
5.4 AI Service development	44
6 Results	49
6.1 Results evaluation.....	49
6.1.1 Testing the prototype workflow	49
6.1.2 Accuracy comparison evaluation	51
6.2 Feedback and the future of the project	54
7 Summary.....	55
References	56
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis	58
Appendix 2 - Final result of description prompt	59
Appendix 3 – Differences between GPT-4 Turbo and GPT-3.5 Turbo company description responses.....	60
Appendix 4 – List of processes provided by Getpart	61
Appendix 5 – Prompts for retrieving processes.....	62
Appendix 6 – Processes comparison prompt.....	63
Appendix 7 – Test results appeared on Getpart website	64

List of figures

Figure 1. Simple list structure example using HTML tags	20
Figure 2. The current architecture of the Getpart registration process	25
Figure 3. Difference in retrieved processes data over 25 iterations	30
Figure 4. Application prototype structure	39
Figure 5. The example response from Aggregation service	42
Figure 6. Example of address object structure	43
Figure 7. Open Data Service structure.....	44
Figure 8. Simplified AI service structure	48
Figure 9. Aggregation service output	51

List of tables

Table 1. Pros and cons of various web scraping services.....	19
Table 2. Overview of Gemini 1.5 Pro API price and throughput options.....	23
Table 3. Overview of different responses of processes data using GPT-4 Turbo	52

1 Introduction

In the modern world, there are plenty of different Internet businesses that depend on receiving and processing unstructured data. One of them is Getpart OÜ, which is facing a problem with a shortage of supplier companies on its platform.

Getpart is a platform for manufacturing on demand. It helps customers search for the proper manufacturer, providing transparent and quality procurement services for businesses. It mainly operates in a manufacturing area. The requirements from the customer and the purpose of this thesis are to provide an automated process for expanding a database of suppliers in the application, to make Getpart more competitive in the market, and to eliminate the problem where companies are lazy to register themselves manually to the application.

An example of getting unstructured data might be web scraping. All the websites are unique, writing a generic script that can parse all the relevant information from a huge number of sites can be either difficult or impossible. It all depends on initial conditions, either the layout of all sites is similar and it's possible to write a tag-based scraper or the number of sites to process is small. In other case it's impossible to use a tag-based approach.

Nowadays, almost everyone has heard of AI (*Artificial intelligence*) and in particular chat bots based on LLM (*Large language model*), which can understand human commands and perform actions that the user asks by writing a prompt [1].

Large language models can help in creating a generic non-tag-based script for scraping information from most sites by sending it the webpage source code for analysis and asking it to find certain information on request using prompts.

This thesis aims to analyze the possibility and further develop a mechanism for finding, receiving, and further aggregating the received data into the database for subsequent use using LLM from OpenAI called ChatGPT (*Generative Pre-Trained Transformers*).

The result will be a working prototype of the application product, that will contain multiple services with interaction with different APIs. The ChatGPT-based AI service is the most technologically advanced and complicated to develop service, that will use pre-defined data from the e-Business Register, sending it to the ChatGPT API for analysis and return a relevant company profile data from a huge number of completely different websites for further aggregation into the Getpart back-end server.

2 Problem statement and purpose of the project

This section describes thesis topic and its relevance as part of the problem statement, presents initial conditions that form the basis for this work, explains the purposes to be achieved and the methodology on which project analysis and development are based.

2.1 Problem statement

Current company registration into the Getpart application is non-automated. First step in the registration process is to find the client's company (further referred to as the client), it means either client finds the application by himself or receives an invitation from Getpart to join. Then a potential client must manually fill out the registration form and wait for manual verification by administrators. This process is inconvenient and time-consuming but can be automated for better performance.

Since e-Business Register provides additional information about companies, it's possible to expand Getpart company's database to show potential customers more detailed information about the preferred company. This will also require some specific changes to the application's existing functionality to bring everything to one standard, based on the latest requirements.

The more complex problem is to have the full picture of information about the company. Since manually registered companies are created by a client, then all the information is presented by the client. However, scraping information from the companies' websites is a saturated problem for the automation due to the necessity of retrieving unstructured data. And this problem is not only local for Getpart, collecting and converting unstructured data to the structured format might be a challenge for a high number of applications [2].

One of the hardest parts is to be able to provide all the relevant information about the company. Almost all the information comes from the e-Business Register API (*Application programming interface*) except for a few of the most important parts, such as: company processes, supported languages and general description. This information

can be found on the company website, but there are many unique websites to process. All of them with different layouts, content-loading approaches, etc. Building a generic website crawling script without using any AI tools for those purposes is too time-consuming, its development may take more time than allocated for writing a thesis. Also, this method can cause various problems associated with the validation of information, since the same information can be written differently or completely incomprehensible. In this case, the use of LLM can replace manual script writing to search for the necessary information, by providing the LLM with a request with a specific prompts that will contain a set of commands explaining what the AI model should do and the context in which it must search for information. The main benefit of using LLM is that AI models based on LLMs are trained of an enormous amount of data and they can understand unclearly written information, as well as analyze and provide an answer based on pre-written instructions.

2.2 Initial conditions

RIK (*Centre of Registers and Information Systems*) provides an e-Business Register service. The e-Business Register is a registry of all legal persons registered in Estonia. It provides an Open Data API with free-of-charge usage conditions that contain about 16 different available endpoints and daily updated downloadable files with all the possible companies registered in Estonia. The Open Data API uses SOAP (*Simple Object Access Protocol*) and WSDL (*Web Services Description Language*) as a data exchange approach. The output from those endpoints is customizable, it can be in a format XML (*Extensible Markup Language*) or JSON (*JavaScript Object Notation*), in the case of this thesis JSON format is selected. To be able to process all companies from the register, the **General Data** archive is used from the **Downloading open data** section in a format of archived JSON file.

All this data is either public or the agreement was signed, so there should not be any ethical issues with it.

Getpart provides a B2B (*business to business*) service, so in case to use this service, the client, either customer or supplier company, must register as a company. If a client only wants to search for a supplier there is no need for any additional actions from its side, however, if the client wants to be a supplier it should fill in additional information about

what processes this company provides, supported languages, lead time, and payment terms. A client as a customer could create a price request for a supplier and as a supplier client could create an offer to a price request.

Getpart back-end written in expanded version of JavaScript programming language called TypeScript using a Node.js runtime, the database is PostgreSQL and using GraphQL as a query language for the API.

2.3 Purpose

The purposes of thesis are:

- Analyze the possibility of using LLM as a generic solution for unstructured data scraping tool for aggregation of company profile data into the Getpart application.
- Develop an application prototype based on ready-made analysis.

The most important part is to analyze the possibility to use LLM and create a separate ChatGPT based service to scrape unstructured data. Choose the most suitable LLM to use.

Before that, a service for interaction with e-Business Register APIs should be built to get relevant structured company data. URL (*Uniform Resource Locator*) must be extracted from that data and sent to the AI service, which will find the rest of the information from the website. The aggregation service should be able to gather all this data and aggregate into the Getpart database using Getpart registration API.

2.4 Methodology

This thesis deals with a principles of design science methodology. This approach is aimed to address and solve practical problems through the analysis, concept generation, prototype development, testing, and optimization [3].

The first step is to analyze the market and find a similar solution that already exists, if there are solutions that can be used to solve the requirements there is no sense in building your solution. The analytical part needs to make sure not to create something that already exists. However, there are plenty of other different aspects that also should be analyzed

such as which technical tools to use or approaches for implementation of the tasks before starting to generate the concept. Last, but not least is setting up business requirements, e.g. budget and deadlines during the analysis stage to ensure the provided resources to complete the project.

The concept generation allows to prepare the basis of the project before actual prototype development. It shows how the project will look like and how it will be used. The project architecture, the communication before different services, and different options for problem-solving are some of the most important parts for preparation before implementing the actual solution.

Development is the most complicated and time-consuming process of problem-solving. This process is iterative and is one of the core principles of design science methodology. It means that if the problems or required changes occur during the development cycle, they should be evaluated and then analyzed again, reproducing the design science development cycle again. Some previously established requirements or decisions can be changed throughout the development stage due to limitations that were not figured out before. The prototype development implements all the goals and requirements from the previous steps, using the architecture that was described in the concept generation phase and considers all aspects that were analyzed in the analytical part.

All solutions, regardless of scope, must be tested, so software development is not an exception. When the prototype is done, all the different corner cases should be tested, the maximum code coverage should be done to avoid situations when the code execution might crash. In the case of this thesis, this is especially crucial due to the usage of ChatGPT API, when every request to the API costs money.

As well as testing, optimization affects the final result very significantly. Badly optimized code might affect both performance and the final cost for this project.

At the thesis's end, the author evaluates how well the final result solves the problem.

3 Problem analysis

This section describes the analysis of the problem. The problems that are going to be faced using different approaches, the advantages, and disadvantages of each of them. The overview of possible existing solutions on the market and its overview with the examples of use cases and their limitations.

3.1 Related solutions

The importance of existing solutions is crucial. If the idea has already been implemented by someone and released for public usage, so there is no sense in making your solution instead of using an existing one. However, due to the recent popularity of AI, there are not many relevant publicly available solutions for this topic, but some of them will be described below.

3.1.1 Open-Source solutions overview

Some open-source solutions might be under the e.g. MIT Licence and modifying or using them is not prohibited. In the case of this thesis, there is only one similar solution was found [4], but tests showed that it can't perform well with current project requirements.

It might be surprising, but there are not many of those kinds of solutions in open access. That was the only most relevant and customizable solution, others provided a more usual approach like parsing by specified tags or keywords.

The only founded solution is called **gpt-automated-web-scraper** and its idea is to create an abstract code, based on the provided information which should potentially be able to scrape the webpage [4]. This solution is not good, after a few tests it was discovered that the script could process only very primitive pages where information is marked up explicitly or even if data is structured, e.g. in tables.

The ChatGPT part of this solution represents only abstract code generation based on a prompt, due to the unavailability of any ChatGPT models to connect to the internet. The author of this scraper predefines some code base and Python libraries along with a prompt

that the user provides when sending a request to ChatGPT. Received code might not always be working, because ChatGPT only tries to build a scraper from the presented information, it doesn't know the exact structure of the page. After a few tests of a webpage where even a description of the company was not marked up as a description, ChatGPT returned a not working code. On the other hand, if the presented webpage has a structure where all the relevant information is marked up explicitly, this approach could be used because consumes fewer resources and will cost less money. This script could be the basis of the development of the thesis solution if it had a better rate of returning the working code. Even though this script only works with 1 page, if it worked well, optimizing it to find relevant information on other pages of the website would not be a problem.

In conclusion, **gpt-automated-web-scraper** has its pros and cons, but the cons are more, so that it may not fully meet all requirements. Especially for processing hundreds or thousands of unique pages, due to its unstable behavior.

3.1.2 Web services overview

There are a lot of different services that provide either AI-based or regular programmed solutions that can scrape information from the website. Many of them are very powerful in different ways, some of them can scrape information from the page by writing a special prompt, which is what is needed for this thesis, or monitor changes on the page.

Table 1 shows a few of the most relevant web services for website scraping that meet the requirements of this thesis, but have significant drawbacks that prevent their use.

Table 1. Pros and cons of various web scraping services

Name of the service	Pros	Cons
Browse AI	<ol style="list-style-type: none"> 1. Capable of training a scraper for a specific webpage. 2. Pre-trained scraper (named Robot) could monitor changes on the webpage. 3. Good choice for a small number of websites with a known location of all information where to scrap. 	<ol style="list-style-type: none"> 1. Requires creating robots for every webpage. 2. Requires specifying the path where to search. 3. Cannot understand if the information is relevant or not.
Apify/Cheerio Scraper	<ol style="list-style-type: none"> 1. Capable of crawling websites entirely. 2. Could find links to other webpages. 3. If the dataset of URLs is prepared, could handle all of them without creating a separate scraper for every website. 	<ol style="list-style-type: none"> 1. Requires explicit path to the content location (excluding link tags). 2. Cannot understand if the information is relevant or not.

All of them have a huge disadvantage compared to the prototype of this thesis, they are only capable of scraping if a specific path to the content location is provided, e.g. html-tag. For example, **Cheerio Scraper** can find links to other pages by itself, but this feature is not customizable. Services like that, without using either NLP, LLM, or any other AI tool could only scrape information, but not understand where to search or decide if it is relevant or not.

All websites are unique, some of them could be developed by novice developers or students without much experience. Due to the niche nature of these companies, the likelihood that the websites were developed by less experienced developers is higher, many companies do not care much about the sites. In these cases, for example, the list of services may be presented in plain text instead of using an HTML (*HyperText Markup Language*) tags that define a list, and the above services may not even recognize where

the information is located. Using AI to analyze content helps to understand where relevant information is and scrape it.

Example of HTML list structure described in Figure 1 below:

```
<ul>
  <li>CNC turning</li>
  <li>Sawing</li>
</ul>
```

Figure 1. Simple list structure example using HTML tags

3.2 AI language model choice

Based on one of the goals of the thesis to find an innovative method for solving problems such as scraping unstructured data, it needs to find an AI model that can meaningfully recognize human-written prompts and generate an answer based on it. The two most suitable solutions for this are to use either LLM or NLP. Models based on LLM and NLP can handle such tasks, but one may wonder which option is best to use.

Both have their pros and cons, but here are the reasons why it is better to use LLM instead of NLP for scraping unstructured data from websites [5]:

- LLMs handle ambiguity better than NLP: Unstructured data often lack clear patterns and can be messy. LLM's, trained on massive amounts of text data, can handle ambiguity, and make sense of the context to identify relevant information. On the other hand, NLP models relying on pre-defined rules might struggle with unexpected variations [6].
- LLMs can understand the intent: Users can express desired data points in natural language. The LLM, understanding the intent behind user requests, can find the relevant information even if it's not explicitly labeled. NLP models might require specific keywords or predefined data point identifiers [6].

3.3 LLM based AI model comparison

Large language models are the hearts of any popular AI chatbots nowadays. This is the technology that helps a machine understand and process human commands. Large language models are called large because they are trained on billions and billions of data

from the internet. To have a local LLM it should be trained in a huge amount of data that will cost either millions or hundreds of millions of dollars due to several factors.

- To buy a relevant GPU (*Graphics processing unit*) (e.g. NVIDIA A100 Ampere) that can train LLM costs around €11000 [7].
- Electricity consumption will also be very height due to the amount of time and power for GPU to process billions of data.

The examples above show that rather than building and training your LLM, it's much easier to pay for a service from a company that has its own LLM and provides APIs for its usage.

Nowadays there are not that many companies that have their own powerful publicly available LLMs to use. The pioneer of the nowadays AI popularity is OpenAI with its chatbot called ChatGPT, which is under the hood at the moment of writing the GPT-3.5 Turbo model. OpenAI provides also a little price access to the API of its latest ChatGPT versions. GPT-4 Turbo model is the most powerful and latest version of ChatGPT at the moment of writing and this version was selected for this thesis.

ChatGPT from OpenAI is not the only one LLM-based solution, giant technological companies such as Google or Meta also have their versions. In the case of Meta, it has Llama, the only LLM solution that can be downloadable for personal usage on your local machine.

Below will be described existing LLM-based solutions in the market at the moment of thesis writing. Their advantages and disadvantages, prices, and capabilities of each. There are 3 of the most popular and trained LLMs from a giant technological company such as OpenAI, Google, and Meta. Describes people's feedback and why a solution from OpenAI is chosen.

3.3.1 Meta's Llama overview

Llama is an open-source LLM from Meta released in 2023 the latest version of which is called Llama 2 (further referred to as Llama). Llama is the most unique and the most different solution due to its possibility to clone it to your local machine, because of its open-source policy. Llama is free to use for most people, Meta does not charge any money

for personal or small business usage. But the quality of the Llama might raise questions, based on public information, the Llama comes in three sizes: 7 billion, 13 billion, and 70 billion parameters [8]. The number of parameters in a model generally correlates with its performance and accuracy [9]. The best advantage of Llama is free-to-use and open-source policy, it allows one to get familiar with how it works and modify the solution. On the other hand, the lower number of parameters Llama has, the performance is worse than that of competitors and to be able to use it, the running machine should have a very high computing power, especially VRAM. So, to be able to use Llama, it is needed either to spend a lot of money on buying GPUs or buy a cloud solution. So, the final price might be even higher for this project than using API and paying per request.

For example, almost no consumer-targeted GPU can handle the most powerful version of Llama. To use it without performance loses at least 35 GB of VRAM, which the GPU should have. The most powerful now is the Nvidia 4090 which has 24 GB of VRAM and costs around €2049 in Estonia. There are possibilities to run Llama on these graphics cards, but the performance will suffer. In the example of the Nvidia 4090 graphic card, the Llama model with 70 billion parameters could fit into 2 consumer GPUs, based on quantized Llama 2 70B to 4-bit precision, there is still needed 35 GB of VRAM (70 billion * 0.5 bytes). The same mathematics, but with different quantization could reduce precision to 3-bit or 2-bit, but then the precision of the LLM will decrease significantly [10].

The price of using Llama in the cloud, e.g. in Microsoft Azure monetizing as pay-as-you-go costs around \$0.00154 per 1000 input tokens and \$0.00177 per 1000 output tokens. It's quite cheap compared with other solutions, but based on the trained data number, other people's feedback [11], performance, personal experience, and complexity with the installation and configuration of all of this compared with using separate API, the Meta's Llama is losing towards OpenAI and Google solutions.

3.3.2 Google's Gemini overview

Gemini (previously called Bard) is an LLM-based AI model from Google that was trained on their LLM called LaMDA. Initially released as Bard on 21.03.2023, later the same year 06.12.2023 it was renamed to Gemini with a massive update to be a real competitor for OpenAI's ChatGPT. Gemini was introduced with 3 different versions: Gemini Ultra, Gemini Pro, and Gemini Nano. Every version has its advantages in a different range of

tasks. Google presented it to be a direct competitor to ChatGPT and on every slide Google compares its model with the GPT-4 model from OpenAI. Unlike Meta's Llama, Gemini has its API for personal usage like ChatGPT, so it opens up an easy way for integration into the applications.

One of the main advantages of Gemini is that the 1.5 Pro version has a 1 million tokens context window that allows one to have a massive conversation without getting out of range. Gemini has the highest of any language model context window so far. It is currently almost eight times higher than the GPT-4 model. The most powerful available version now is Gemini 1.5 Pro, but even with its weaker version Gemini 1.0 Pro, they are still available only as a "Free of charge" with tons of limitations. Table 1 describes a comparison between the **Free-of-charge** and **Pay-as-you-go** options of Gemini 1.5 Pro, the only publicly available at the moment version of Gemini.

Table 2. Overview of Gemini 1.5 Pro API price and throughput options.

Description	Free of charge	Pay-as-you-go
Rate Limits	2 RPM (requests per minute) 32,000 TPM (tokens per minute) 50 RPD (requests per day)	5 RPM (requests per minute) 10 million TPM (tokens per minute) 2,000 RPD (requests per day)
Price (input)	Free of charge	\$7 / 1 million tokens (preview pricing)
Price (output)	Free of charge	\$21 / 1 million tokens (preview pricing)
Availability	Now	Coming on 02.05.2024

The prices and limitations information are taken from the Google AI for Developers website.

From a personal user experience using the mid-tier version used in the Gemini chatbot, it feels like ChatGPT understands the user's prompts better and generates more relevant responses even if a prompt is written poorly. Google doesn't provide which exact version is used in the Gemini chatbot.

But unlike ChatGPT, the Gemini API and Google AI studio are unavailable in Estonia and in the EU (*European Union*) at the moment of writing. So, the only AI model to use left ChatGPT from OpenAI.

3.3.3 OpenAI's ChatGPT overview

ChatGPT from OpenAI was the first AI model that popularized solutions based on LLMs for the whole world. Even though the release of GPT-1 was in June of 2018, the most popular became version 3.5 with the release of chatbot called ChatGPT on 30.11.2022. From that day popularity of AI for a huge amount of people became obvious, and every giant technological company wanted to have their version of AI models for a rapidly developing newly formed market. The most capable version became the GPT-4 model that was released on 14.03.2023 and from that point, it remains the newest version, except only the GPT-4 Turbo model, but for now, it's not classified as a separate version but it has a better understanding of users input, bigger context window with 128 thousand tokens and the lowest prices per tokens in a GPT-4 family.

The number of users of ChatGPT, either ChatGPT as a bot or API, is huge. The community is massive, in this case, using products from OpenAI is better due to the amount of people that can help. OpenAI provides a developer's forum, where everyone can ask questions, answer problems, etc. This is a huge plus for using OpenAI products, even though it might not be an initially obvious reason, but if something goes wrong, people in the community can help or the problem is already being discovered and solved.

3.4 Current implementation problems

The current implementation has several problems that make registering new companies and expanding the supplier's database inconvenient and time-consuming.

The current registration flow allows to addition of new companies to the application only by submitting a special form either by admins or clients that have been invited to the app. Filling a registration form by a client is a kind of already legacy approach in this application, due to its inconsistency for potential clients, many of those do not want to do it by themselves or even some think that this idea is pointless with this kind of flow. So, with that approach, after filling in the registration form, manual verification should be done by the administrators. The precursor of this thesis idea is the possibility of adding

new companies by administrators manually so that created companies are matched as **faceless**, which means that this company is not related to any user. It allows Getpart not to contact companies directly, instead using its public data to fill out the form and present it as a supplier at this application. Even if the company would like to register in this application, it can still fill out the registration form, but after the verification process, this client will become the owner of its company.

From all of that it becomes obvious that rather than adding companies one by one using those approaches, which is time-consuming and inconvenient, there is a possibility to expand suppliers' databases using publicly available data from the e-Business Register and company's websites by creating a special service that will handle all of this.

The image below shows the user registration flow on the Getpart platform.

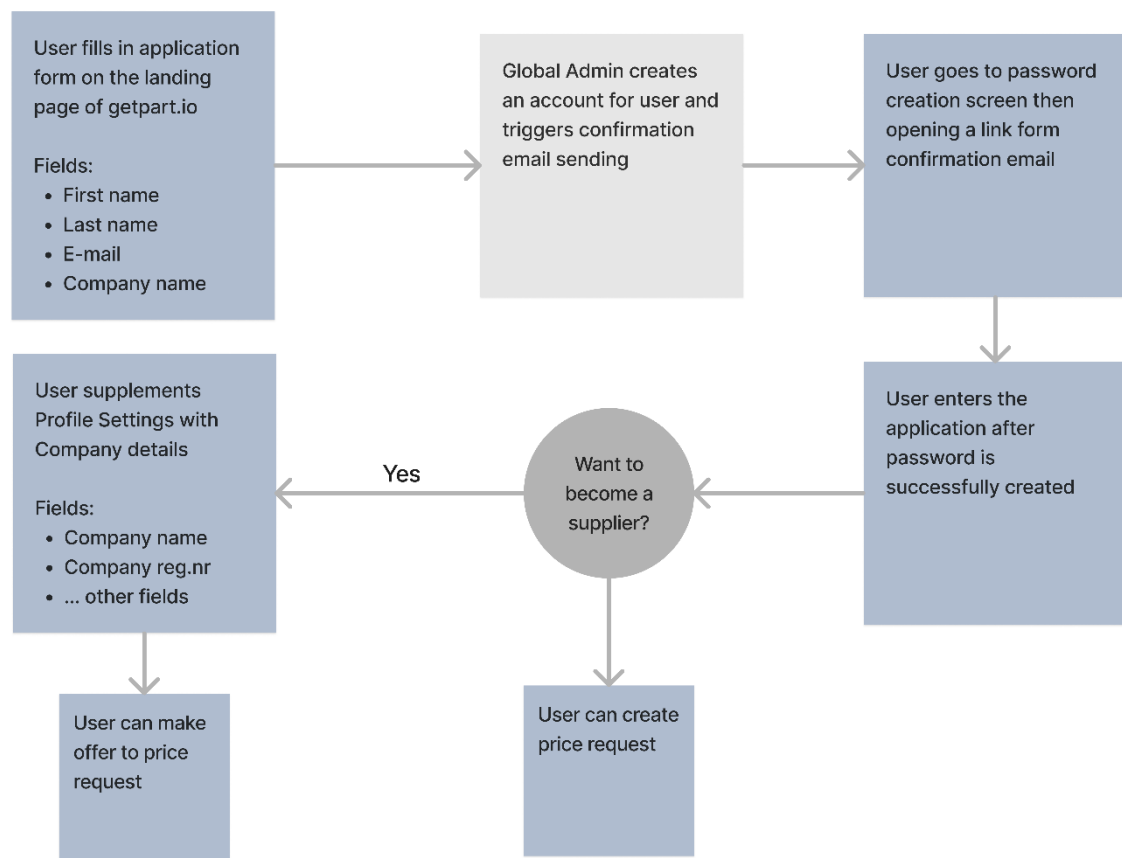


Figure 2. The current architecture of the Getpart registration process

3.5 A vision of the solution

Processing thousands of websites might not be a trivial task. There are many different factors to be considered, starting with the uniqueness of each website in the case of UI (*User Interface*), ending with content rendering and location of the necessary information in the DOM (*Document Object Model*) tree.

Even though the process preceding in the earlier described chapter requires only investigation in Open Data API of e-Business Register, the building of a parser that is capable of handling the information about up to hundreds and thousands of companies. Due to the using of Estonian Government Web-Services there is not a single publicly available solution that can get full information not only about the general data of the company but also some specific data that e-Business Register does not provide:

- Information about the processes that the company provides.
- Supported languages.
- General description.

The solution being created will allow Getpart to combine the company's information from different sources and present it to the potential client. Previously described missing information in the e-Business Register might be found on these companies' websites. So, for this information search the ChatGPT-based AI solution is a very useful tool due to the number of websites to process, their complexity, and their uniqueness.

At the beginning creating a solution might remember existing websites in Estonia that provide general information about the companies, such as **teatmik.ee**, **e-krediidiinfo.ee** or **inforegister.ee**. But unlike those sites, Getpart is aiming to provide a service to connect a client with a supplier, so for a better understanding of who this exact supplier is, it was decided to provide additional company information for that client, so the client does not have to leave Getpart site and go to like e.g. **teatmik.ee** to find this information.

The most important and the most powerful part of creating a solution is an AI service. This AI service will be able to get relevant information by searching it throughout the whole website, the only thing that is needed is the website URL. Unlike the similar solutions that are being discovered and described in one of the previous chapters (see

chapter 3.1), this solution does not require either a keyword or a specified area in the DOM tree where to search. The only thing needed is the source code of the page and precisely well-described prompts. Prompts are pre-defined in the application, but the source code is always pre-loaded before sending a request to a ChatGPT API. To be able to get any information from the website, the AI service will require at least 3 requests to ChatGPT to get relevant data in case of a one-page primitive website. If the data won't be retrieved during the first iteration, so every other iteration will also require a minimum of 2 requests to the ChatGPT API and this loop will end only if the needed information is either found or there are no pages left where to find the information. So, this solution is being created only for specific purposes and it's not editable in terms of changing requests to which information to search.

4 Solution analysis

This section defines the requirements for the solution prototype based on the vision of the solution, explains the restrictions of the prototype, describes the technologies chosen by the author, and compares various prompts in AI service.

4.1 Solution requirements

The main requirement for this solution is to verify that AI models based on LLM could be used for scraping unstructured data, in the case of this thesis from the huge amount of the websites. Around that building an application prototype that could use retrieved company's profile data and aggregate it into the main Getpart application. At the same time, it needs to be considered that the thesis is strictly limited by time, so building an AI service for retrieving the information from the websites and surrounding services to be able to run AI service are the main requirements. The microservices approach will be used in this project to organize independent service communication between different APIs.

4.1.1 Functional requirements

The created solution must enable the following functional requirements:

1. The application must be able to receive a request from the Getpart registration API and start the application.
2. The application must be able to download general data about all existing companies in Estonia from the e-Business Register.
3. The application must be able to sort downloaded data by specific categories, and existing and working websites.
4. The application must be able to get a website URL from the sorted data and send a request to AI service for processing the website.

5. The application must be able to get missing information from the AI service and send it back.
6. The Aggregation service must be able to combine all the information from different services, process it, and aggregate it to the Getpart back-end.

4.1.2 Non-functional requirements

The created solution must support the following non-functional requirements:

1. The application must be stable enough to handle hundreds and thousands of company data.
2. The application must not crash if an error occurs, the problematic company should be noted, and the workflow will continue.

4.2 Restrictions on the prototype

Due to the thesis writing time limitation, the response's accuracy from ChatGPT will also be lower than expected. It does not mean that invalid data will be received, but the answer might not be full.

4.2.1 Incomplete data return

Using the example of 25 test runs of the **Hismekano Estonia OÜ** website was discovered that instead of receiving invalid or artificially generated data about the processes, it just returned incomplete data. These tests were done using the “gpt-4-0125-preview” model, it’s the newest version of gpt-4 turbo at the moment of writing a thesis. The GPT-4 version is used instead of GPT-3.5 to minimize the **hallucination effect** in ChatGPT response. The hallucinations in ChatGPT are artificially generated data [12]. In the case of using GPT-4 Turbo, after the 25-test run, there was no artificially generated data identified, but around 60% of retrieved processes data was not full. It only contained 75% of the possible relevant information that could be found on that page.

The graph below shows the number of different processes returned from 25 iterations with GPT-4 Turbo. As can be seen in the graph, the results when process **Sawing** was

returned is 40% from 25 iterations. The valid result here is **Assembly, CNC Milling, CNC Turning, and Sawing**. Nevertheless, even in an incomplete answer, which results in 60% of all returned responses, 80% of the information returned was valid.

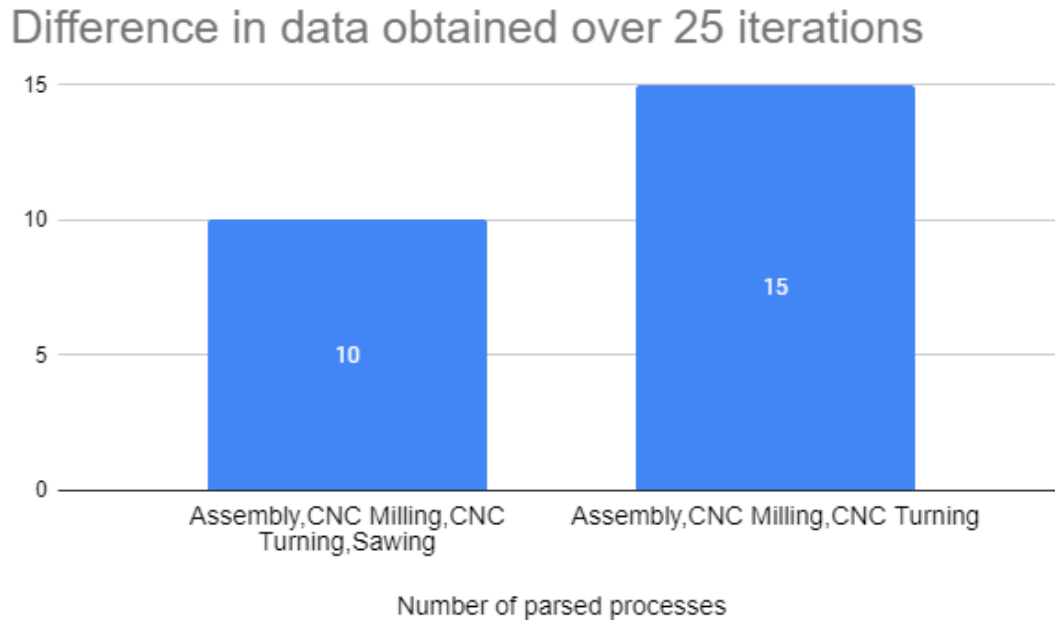


Figure 3. Difference in retrieved processes data over 25 iterations

4.3 Technology choice

The choice of technologies to solve the problems posed comes from many factors, directly one of them, is the selection of an AI model, which was already presented in one of the previous chapters (see chapter 3.2).

It is also worth mentioning that the main application has already been created and for potentially new employees, if possible, it is worth using the programming language that is already used in the company, but only if this does not interfere with the writing of the application product. At the moment, Getpart uses Node.js at back-end, it has also many advantages for solving project requirements [13]:

- **Event-Driven & Non-Blocking I/O:** Node.js operates on an event-driven, non-blocking I/O model, which is efficient for handling large I/O operations like reading a huge JSON file in this project without blocking the main thread.

- Caching: The Node.js caching feature can improve performance by storing processed data in memory, reducing the need to reprocess the same data.
- Cross-Platform Support: Node.js application can run on multiple platforms, which means the parsing service can be executed on various operating systems without modification. It is extremely crucial due to the current development on Windows and further application containerization for deployment to AWS.

Those were some of the main advantages of Node.js for solving problems in this project. But of course, tons of different programming languages could do the same.

Unlike using other programming languages to solve current problems, there is no need to saturate the entire Getpart infrastructure with many different programming languages for no reason and make the infrastructure garbage out of it. Even if Node.js could perform a bit worse than e.g. Java, C#, or Python.

Additionally, it is important to mention that Node.js is not a framework or programming language it's a JavaScript runtime [14], due to that fact it can use tons of libraries from npm (*Node Package Manager*). Npm contains hundreds of thousands of different libraries for almost everything.

On the other hand, deciding which programming language and tools to choose for AI service development. Due to the high number of different libraries and popularity of Python for AI and ML, it was decided to choose Python for building a separate AI service.

Also, it is crucial to have a separate server due to the microservice architecture of the whole project for building independent separate APIs [15].

Before moving on to an overview of the libraries used in each service, it is important to note several task-independent tools that will be used by the author everywhere:

- IDE: There are tons of various IDE's nowadays, starting from a lightweight Sublime Text or Visual Studio Code, ending with full-blown IDEs like JetBrains IntelliJ IDEA, Visual Studio etc. JetBrains usually provides many various IDEs for different tools and programming languages, for example for JavaScript development WebStorm, or PyCharm for Python development. However, IntelliJ IDEA provides a ton of plugins for supporting different programming languages,

syntaxes, etc., so instead of using multiple IDEs for every Node.js or Python services, like WebStorm or PyCharm, it was decided to use IntelliJ IDEA for this project development due to its flexibility.

- **Software Packaging:** Not only for the microservices architecture of the project but also for the application deployment application should be packaged. Software packaging or containerization packages an application and all its dependencies for consistent execution across different environments. In this project decided to use Docker as a tool for packaging, it's the most popular nowadays solution for these purposes. Docker is more comfortable and understandable for the author.
- **VCS:** There are many different VCS on the market right now. Bitbucket, GitLab, or GitHub are the most popular and affordable solutions. Due to the existence of the Getpart application on GitHub, it was decided to continue using GitHub as a version control system in this project for publishing the source code of the application.

4.3.1 Open Data Service overview

One of the libraries that will be used in this project is Express.js, a minimalistic and fast framework built on Node.js that provides many useful and time-saving features to help set up a server and manage HTTP (*HyperText Transfer Protocol*) requests and responses [16]. There is no need to use more heavy tools such as Nest.js or Next.js due to the limited functionality of the created solution. The lightweight Express.js is better suited for project requirements.

There won't be a separate database for this project, every iteration of this application prototype will call Getpart back-end API and save a newly created company in the existing application database, the database used in the back-end is PostgreSQL.

One of the main purposes of this service is data exchange with government services by using the SOAP approach. Government services like e-Business Register provide an Open Data API for getting companies' data by sending XML requests to an e-Business Register WSDL service. This WSDL service file provides all the possible XSD (*XML Schema Definition*) schemas that the e-Business Register has. All the possible information

about creating an XML request is provided in Open Data API documentation or by using software called Soap UI that can parse all the XSD schemas from a WSDL file and prepare an XML file with all the fields out of it. The responses from the Open Data API are customizable, it is possible to switch the response type from XML to JSON or the information language.

On the other hand, for a data exchange inside this project, the REST (*Representational State Transfer*) approach is used. REST is easier to write and it's also faster than SOAP, REST leverages the power of HTTP, and responses to REST requests are provided in smaller text format, e.g. in JSON [17].

For unzipping huge JSON files downloaded from the e-Business Register many JavaScript libraries had a problem in that they cannot handle several gigabytes files. For solving this there were multiple options, either to split the parsing mechanism into chunks to read smaller pieces of a file or to use another library that can have this functionality out of the box. The variant using a more powerful library was chosen, due to the complex optimization and worry about losing data while self-building this chunk-splitting mechanism. A good choice for that is a package from the npm registry called **yauzl**. This package fixed the problem of keeping memory usage under control, previously either built-in libraries or other founded libraries in the npm registry did not provide this functionality.

The big part as can be seen is the I/O operation for handling and processing files that were downloaded from government services. For those purposes, built-in Node.js libraries were used such as fs, URL, path, etc. However, for manipulation with either creating or parsing JSON files, the **JSONStream** package was used for those purposes.

HTTP(s) protocol was used to communicate and exchange data between various application services and the Node.js **axios** library was used to process them.

The different sensitive information, e.g. API secrets or authentication codes aren't stored inside the application, all this data comes from the .env file using **dotenv** library in combination with the **convict** library for configuration string management.

4.3.2 AI Service overview

As was already described at the beginning of chapter 4.3, the Python programming language of version 3.11.8 was chosen for building an AI service. For building an API of AI service the FastAPI web framework is used. FastAPI has very high performance, on par with Node.js, and it is designed to be easy to use and learn. After the main code of the AI service was written, the need came to wrap this code in an API and FastAPI turned out to be the simplest and fastest solution. To be able to use FastAPI in production an ASGI server such as Unicorn was needed.

Before using GPT-4 from OpenAI it was needed to prepare some code and packages for scraping source code from the webpage. For these purposes, here is the list of some packages that were used:

- Requests: It's a simple HTTP library that allows to send HTTP requests extremely easily. The requests library is mostly used only for checking status codes while trying to access the webpage, rather than using it for webpage scraping. For webpage scraping Selenium is used, but while it doesn't provide status checking decided to still use requests for this small task.
- Selenium: Instead of using BeautifulSoup for webpage scraping, Selenium is used since many webpages could contain dynamic content. BeautifulSoup will not be able to capture dynamic content, while Selenium can. Webdriver-manager: To be able to use Selenium for web scraping of dynamic content, webdriver-manager is used for downloading Chromium which can allow to execution of JavaScript code that can be on the webpage.

It is worth mentioning that these were some of the main packages that were needed for web scraping. There are many other packages that help e.g. with any kind of validations or other libraries that are needed only for the main ones to work.

To use OpenAI's ChatGPT API it is needed to install **openai** library. This library allows to make a **conversational** window with a ChatGPT and provides some configuration for changing behavior of the ChatGPT. Also to be able to track and calculate the prices for the requests and changes in some places a ChatGPT model version was needed to install a package called tiktoken. The tiktoken is a BPE tokenizer for use with OpenAI's models

[18]. The main purpose of using tiktoken in this project is only to count tokens in a prompt, it is crucial due to the different prices per request to models. Sometimes it is better to use e.g. GPT-3.5 for smaller tasks that do not require high precision, thereby saving money.

4.4 Various prompts comparison

One of the most important parts of this thesis is to understand if it's possible to use LLM for scraping unstructured data, and the answer is yes. However, it's important to prepare an excellent prompt for the AI model to force it to return the most relevant response [19].

As was discovered from building a prototype, the more highlighting of a particular query in a prompt by using, for example, exclamation marks or by making whole words in upper-case, the better. It has more effect on a GPT-3.5 model rather than on GPT-4 Turbo, but even GPT-4 Turbo while processing complicated sites can misunderstand the prompt without highlighting.

4.4.1 Retrieving company description data

Appendix 2 contains a prompt for requesting a company description and summarizing it. As can be seen, highlighting is not everywhere, only the most important parts are highlighted using either many exclamation marks before and after the sentence or by highlighting the whole word or the sentence by writing it in upper case. In other words, need to somehow highlight the most important information so that it stands out from the rest of the text. For a test run chosen **Hissmekano Estonia OÜ** website, doesn't have a complex structure, but at the same time, it's not a way to primitive. Using this prompt with a GPT-4 Turbo (**gpt-4-0125-preview** the latest version) this AI Service finds and scrapes the information in 3 iterations of the recursive function, on the other hand, GPT-3.5 Turbo (**gpt-3.5-turbo-0125** the latest version) processed it in 13 iterations of the function. However, the received result was not valid due to issues with a making summary of the company description.

Appendix 3 shows both description outputs of GPT-4 Turbo and GPT-3.5 Turbo. As can be seen from Appendix 3, the description results from GPT-3.5 Turbo is just some kind of description where the company information might be found and a link to this page, on

the other hand, GPT-4 Turbo returned a well-written decent summary of the company description.

As was said before, only 3 iterations were required to process the company using GPT-4 Turbo, however GPT-3.5 Turbo using the same prompt might not return even invalid data in 13 iterations, for the next few tries it took more than 25 iterations for processing the company data, and the process continued to call the recursive function again and again.

4.4.2 Comparing more sensitive data

The more sensitive data in the case of this project is provided by the company processes. This information might be written differently on different websites, e.g. nowadays almost every process, such as turning, milling, or bending is CNC (*Computer numerical control*) based. However, existing names of those processes in the Getpart database might be different, i.e. do not contain a CNC prefix before the process name, or process names on the company's websites might be described too abstractly. Since the number of companies to be processed is enormous, the prompt must include an explanation for those cases when data is unclear. Appendix 4 contains all the processes that exist in the Getpart database and on which the ChatGPT should rely when comparing processes.

As an example, the website chosen **OÜ Istrek** company website, has a quite primitive layout that contains almost all relevant information on the main page, so excluding the right now company description in the **About us** page due to current testing only with processes. Every test run requires 5 attempts, both with highlighting and without highlighting, using also GPT 3.5 Turbo and GPT-4 Turbo, so the total amount of tests was around 20.

The valid processes response has contained: CNC Milling, CNC Turning, Drilling or Drilling/Tripping and Stamping. While GPT-4 Turbo had the same responses with and without highlighting, the valid responses were about 80%, so only 1 returned answer doesn't contain either the Drilling or Drilling/Tapping process. Although GPT-3.5 Turbo was not as bad as expected, 20% of the responses contained the Welding process using highlighted prompts, and about 40% using prompts without highlighting. It might be not that bad, due to the price per request of these models, but the performance and accuracy are more important than the price in the case of this project. In addition, while GPT-3.5

Turbo was used, all the answers when the prompt was not highlighted, contained weird company descriptions, however, there were not any changes in the description prompt.

All this comparison between which prompt is better or worse for different models is very important. The cheaper version of GPT-3.5 Turbo might not be bad, it might surprise with a few requests when it returns a valid response, but further test runs may show the differences in responses. However, the main goal is to have a stable version, that works all the time the same and doesn't change its behavior by returning sometimes artificially (hallucinated) generated data [20].

As can be seen from Appendix 5, even for human eyes the highlighted prompt is easier to read. All the main commands to ChatGPT are highlighted to separate them from the rest of the prompt, those commands are telling the GPT model what to do in unclear situations. It is extremely important for GPT-3.5 Turbo, it is better to understand separated text with different commands rather than the whole complete text. If the GPT-3.5 Turbo were used, Getpart might lose tons of information, because sometimes ChatGPT could not understand the prompt.

4.4.3 Importance in accurately describing the prompt

Since ChatGPT is not training with every other request, it is useless to expect a better performance while sending it new requests, it's not training. However, it might be only trained during the session, which means that it is possible to give it a piece of information while dialogue with ChatGPT models is still going on, for example in this project such information is either predefined lists with processes or formatted webpage source code. It always sends separately from other prompts to pre-train ChatGPT with a piece of information that it should rely on.

While many people might think that prompt writing is easy, they are kind of right, however, to get a decent result especially when the request is huge and requires precision, the prompt must be designed in a way where every step must be described separately and so that even a small child can understand it. This project is always handled with huge prompts, which is why it should be well optimized to not lose the efficiency and capabilities of ChatGPT for handling the request [21]. The user should carefully and accurately describe a task for ChatGPT and, most importantly point at different corner cases that could arise.

Appendix 6 contains a processes comparison prompt. The most important parts, like telling to target the processes list to rely on and how to interact when there is no anything similar, or paying attention to sensitive information, such as language of the processes. Other less important information is writing without any highlighting, this prompt is really good and works fine with both GPT-3.5 Turbo and GPT-4 Turbo.

5 Solution implementation

This section describes the implementation of the prototype, the requirements of which were described in the chapter Solution Analysis (see Chapter 4). It describes a step-by-step implementation of the solution, explaining all the actions from the author.

5.1 Application architecture

The application uses a microservice architecture since many services are almost completely independent of each other. Also, this approach will help improve the efficiency and sustainability of applications in the future [22].

A deeper dive into the architecture of every service and their overviews will be presented in the next chapters. However, Figure 5 represents the whole prototype architecture.

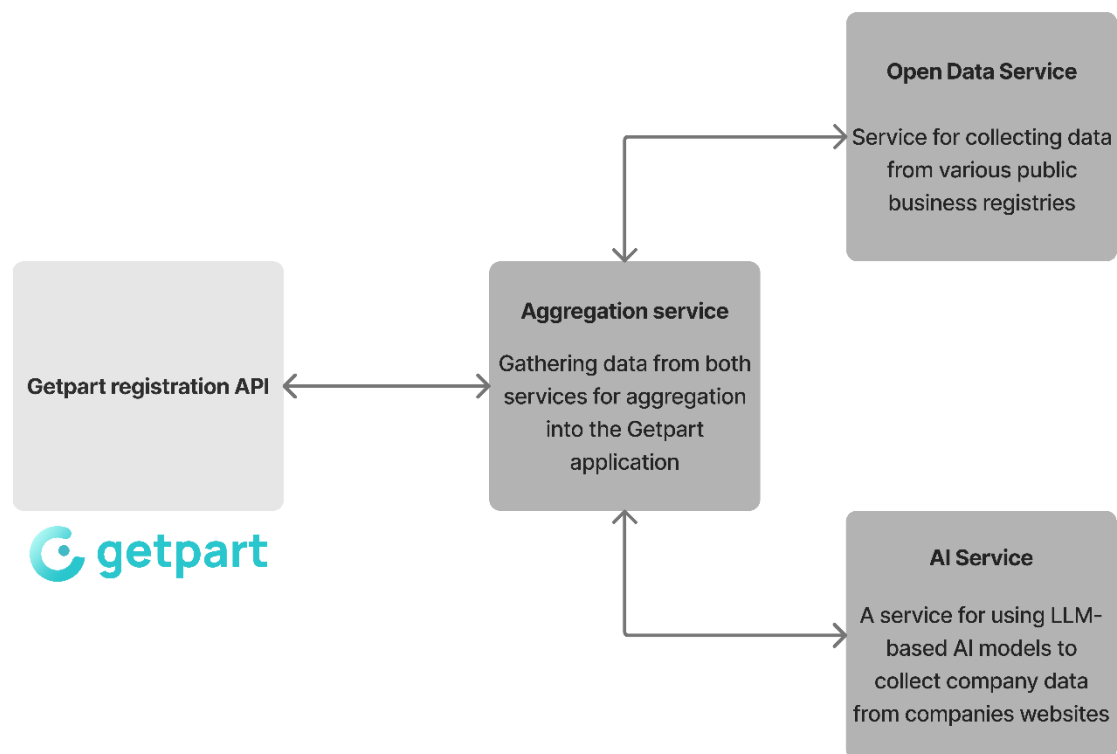


Figure 4. Application prototype structure

The application consists of an **Aggregation service**¹ and 2 separate services called **Open Data Service**² and **AI Service**³. The **Aggregation service** represents the starting point of the application that gathers data from the other 2 services and aggregates it to the Getpart using its registration API. Both **Open Data Service**² and **AI Service**³ represent services for interaction either with open data from the government services or AI services. At this time, they both contain only 1 API, but further development of the application may require interaction with other services in both areas so that every new functionality will be added to a specific service with its API.

Proper dispersion also helps developers not only from the technical side but also from the convenience side, where each service has its specific task.

5.2 Aggregation service development

As was described in previous chapters Node.js, the JavaScript runtime, is chosen for building most of the application services, except the AI service³. One of the main benefits of Node.js is the easy-to-start approach, the JavaScript programming language allows developers to be more flexible in writing code than for example in Java or C#. It brings either a class-based or a functional approach for code writing and application building. However, description of the JavaScript says that it is an object-oriented programming language [23], the solution implementation is done without using classes in JavaScript at all by writing separate scripts for different needs.

The first step in creating a Node.js-based application is by executing the **npm init** command for a project initialization. It will generate the core of the Node.js application the **package.json** file which is the central place for application configuration, dependency management, etc.

The next logical step is to install all the necessary dependencies that will be used in the project using npm:

¹ <https://github.com/allerk/getpart-companies-aggregator/tree/main/aggregator>

² <https://github.com/allerk/getpart-companies-aggregator/tree/main/open-data-service>

³ <https://github.com/allerk/getpart-companies-aggregator/tree/main/ai-service>

- Express.js: It's a minimal and flexible Node.js web application framework that provides a robust set of features for building the API's. Helps easily set up a server and handle HTTP requests and responses.
- Nodemon: An especially useful library for application development. It helps to exclude the need to re-run the server every time to apply changes, Nodemon monitors changes in a project while the server is running and automatically restarts it. To use this library, need to create a new script in package.json that will run the application using Nodemon.
- Axios: Instead of using standard Node.js modules for sending and receiving HTTP requests or responses, it is decided to use Axios due to the easy-to-read syntax with decent provided methods like get, post, etc., and promises-based approach instead of callback. The built-in Node.js modules require more verbose code, involving callback and error handling that can become cumbersome for complex requests.
- Dotenv: Simple module that loads environment variables from a .env file.
- Convict: A configuration module that provides a configuration schema that helps to give more context on each setting and enables validation and early failures.

The structure of the Aggregation service is quite simple, the project starts from an index.js file in the root directory that creates an express server instance for handling the incoming requests. All the business logic is in the **src** directory².

After the request in index.js is received, the endpoint calls the main function of the whole application aggregator.js. This is the place for information gathering from other services and further aggregation into to Getpart registration API. The preparation for aggregation data to the registration API implies a first request to the **Open Data Service** and by receiving and processing that information further request to the **AI service**. Getpart's back-end uses GraphQL as the query language for the APIs, so before preparing GraphQL mutations a few queries have to be sent to the back-end for data comparison and adjustment due to their differences in presented data in Open Data and the Getpart database. When the data is ready, the Aggregation service returns a response to the

Getpart registration API in the format of a number of companies processed with successful and rejected company registry codes which is shown in Figure 5.

```
{
  "processed_companies": 3,
  "successful_ids": [
    234235481,
    539888193
  ],
  "rejected_ids": [
    12122121
  ]
}
```

Figure 5. The example response from Aggregation service

The details of Aggregation service implementation might look weird due to registration logic on the Getpart back-end which was not ready to register companies in this way. However a complete rework or major changes to the registration API logic are out of the scope of this thesis.

5.3 Open Data Service development

As was told at the beginning of chapter 5.2, the Open Data Service is also written using Node.js. Many aspects of the initial preparation for building Node.js application already was described in chapter 5.2, so there is no sense in repeating them here.

All the installed packages for Open Data Service are the same, except for adding package called **yaubl**. This package is incredibly important, it allows to decompress really huge files, many other similar packages tested by the author were unable to process a 3.8-gigabyte zip file containing all the data of companies registered in Estonia. So rather than writing its logic for splitting the unzipping process into chunks and controlling the application memory, the relevant solution was found the on **npm public registry**.

Open Data Service has only one endpoint under the **/api/business-register** path that is calling an e-Business Register parsing logic. However, the further development of this application might need to use other open data services, so the purpose of the Open Data Service is to have multiple endpoints for handling different services. In the case of this thesis, the only open data service is from the Estonian Business Register.

The business logic of the business-register API of the Open Data Service is actually divided into 2 parts. The first one is handling I/O operations by downloading and decompressing the company's archive, on the other hand, the second part is processing received data from the first part by reading it and calling additional e-Business Register APIs to get missing data. Unfortunately, the e-Business Register does not provide all the relevant data in one archive; quite a lot of information is missing there. Because of this, it is necessary to call additional e-Business Register APIs to retrieve missing information, even if the whole response from the API is useless and only one field is valid. Downloading is the first step in this service, the HTTP request is sent to the e-Business Register server to retrieve the archived JSON file with the data of all registered in Estonia companies. When the archived JSON was downloaded, it must be decompressed to be able to use it, for this purpose earlier mentioned `yauzl` package could help here. Rather than loading the archive all at once, `yauzl` open method has a parameter called **lazyEntries** and when it has the value **True**, it loads files on demand which helps the application prevent going out of memory and crashing.

The last step is to parse data from the unarchived JSON file and filter it by specific categories from the EMTAK (*Estonian Classification of Economic Activities*) [23] list, in the case of this thesis companies are sorted by "Machining" category. The **parseData** function is responsible for all those operations. The main criterion before sorting is to verify if the company has a workable website. If a company has a functioning website and the status of the company is **R**, which means registered, then the iteration might be started through the list of categories in which the company operates. When all the filters are passed then several helper functions are run to prepare data by formatting it to the valid format to be in maximum compliance with company data models in the Getpart back-end. Figure 6 shows an example of the formatted address object output.

```
country: 'EE',  
zipcode: address.postiindeks,  
city: city,  
address: address.tanav_maja_korter,  
state: state
```

Figure 6. Example of address object structure

To prepare and format the data, information from the JSON file is not enough. When the helper functions are called, some additional requests to the e-Business Register APIs are

sent. If downloading from the e-Business Register was done using simply GET method, without any specifications, then requesting missing information from the other APIs required for preparing XML requests. For preparing XML requests there are several XML schemas for every needed API from e-Business Register ready. All these schemas contain several options, but the main ones are the credentials, output format, and language. The credentials are received from the special config file that exists in this service using the **convict** library, and the data for the configuration file comes from the .env file. So, all the secret information is secured and not used anywhere in the code directly.

When the data is parsed, the JSON file is deleted to free up space and data is returned as a response to the **Aggregation service** request. Figure 7 describes the structure of the Open Data Service business register API.

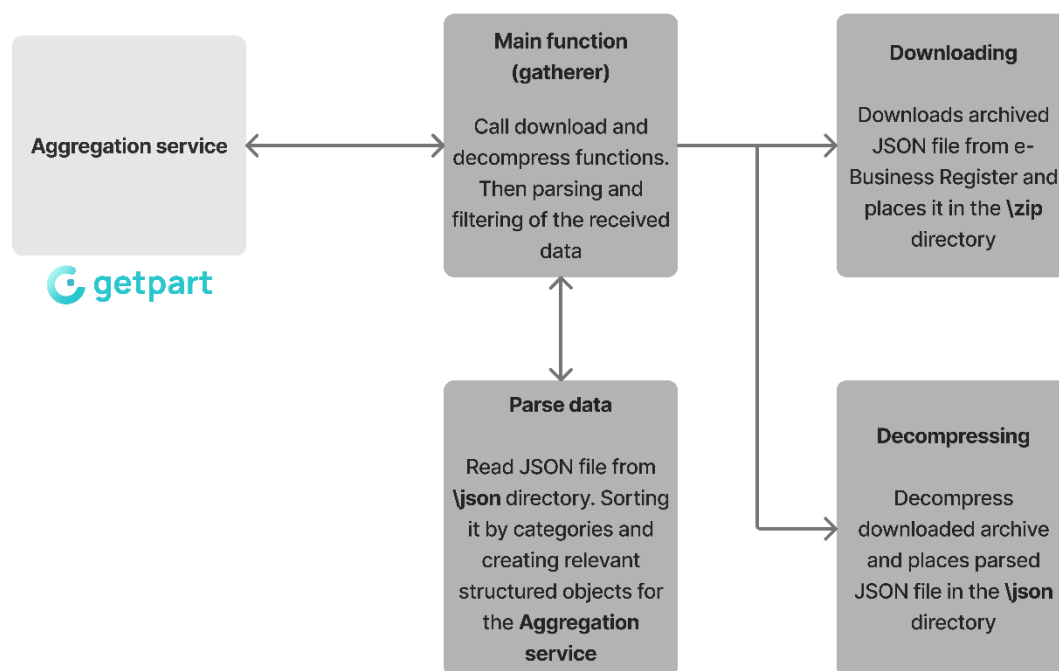


Figure 7. Open Data Service structure

5.4 AI Service development

The development of AI service is different from the previous services. The first difference is using Python programming language, rather than using Node.js, although OpenAI supports using Node.js, it is better to use Python due to the popularity of this programming

language for AI and ML (*Machine learning*). Python is widely used for writing an AI application, it has many more different libraries not only for AI implementation, but also for data analysis than Node.js, and a significant plus for using Python is a larger community that is mostly uses Python for those purposes [25]. Most of the questions that might arise will most likely be answered on different forums using Python.

The first step of building a Python application is to choose an environment manager, in this project, the environment will be Python's standard tool for creating isolated virtual environments **Venv**. Instead of using others, e.g. **conda**, in this context, the standard lightweight and simple **Venv** is more than enough for this project, because this service is only Python-focused, and it doesn't require complex solutions that can require a complex web of dependencies. The **Venv** folder in the root folder of the service describes all the information, either packages or the path to the Python executable file.

The main code is in the app folder, it's important to note that to be able to make a Python file a package and use it in other files need to create an **__init__.py** file that will treat Python that this folder as a package. Since the version of Python 3.3, Python generates them automatically and the content of these files is empty.

File main.py starts the application by using FastAPI as a server builder framework. The work of this framework is the same as in Express.js. At this moment, the only one endpoint that the AI service has is to use ChatGPT API under the **/url/{url:path}** endpoint path.

For handling different exceptions from the OpenAI ChatGPT API and not only, specially created a generic **GlobalError** class for handling processing the error to make a single error structure.

The starting point for handling and processing the ChatGPT responses is in the **webpage_analyzer** package. This package contains all the necessary functions with the creation of a **GptAnalyser** object that communicates with the ChatGPT models. GptAnalyser is a class that contains all methods for different purposes of communication with ChatGPT and the instance with all options of the ChatGPT window. Here is a small description of all fields that is needed for working with an API in the context of this application:

- **Model:** Defines which model of ChatGPT API to use.
- **Temperature:** Defines the temperature of the model, the lower the value the more precise it will be, the higher the value the more creative it is.
- **SYSTEM_MESSAGE:** It's a helper generic prompt to indicate the model is, e.g. "You are an expert in website analyzing and scraping". It will give the model more context in which area it should answer.
- **Possible processes:** The prepared list of processes that the model should rely on when looking for processes/services that the company provides on its website.

GptAnalyser gets those values in the constructor and assigns their values to the desired variable. GptAnalyser has 2 methods, the first one **get_website_language** for retrieving a website language and **get_info_from_a_webpage** for retrieving different information from a webpage.

All the specific logic is separated from each other in different packages, e.g.: prompts creation functions, webpage interaction functions, or different formatters. The `webpage_analyzer` package contains two functions, the first one that starts the web crawling logic and the second that allows it to work until the relevant information is not found by using a recursive approach.

Some of the functions like retrieving webpage source code require flexible configuration. This package works using **Selenium** rather than **BeautifulSoup** for retrieving webpage information even from dynamic webpages. It uses Chrome driver to be able to run JavaScript on the client side to get missing data. This approach requires flexible configuration due to differences in the work of Chrome and Chrome driver on different machines. The author is using **Windows 11**, and on that operating system with the installed GPU on the machine, there is no problem for Selenium to open a browser window with a webpage and execute all the code there. However, when this AI Service is containerized, several issues might arise. One of them is that a virtual machine using docker images doesn't have its own GPU for opening the browser. This problem not only arises when Selenium tries to scrap but before, when the **Webdriver-manager** tries to install **Chrome driver** and prepare **Chrome**. To fix a problem with opening browser window is created with a special configuration function for telling Chrome web driver to

run in a special mode, e.g. by providing these arguments **--headless** and **--disable-gpu**. But also, to exclude installation right in the code, like it is done on the author machine, for the containerized version the Chrome driver is downloaded from the Chrome testing public build because this is the only place to get the relevant latest version of Chrome driver that is supported by the installed Chrome.

When the AI Service is running, all the data is received to run the scraping mechanism, the GptAnalyser starts its work. When script try to find relevant information, firstly try to find an English version of the website to get the best performance out of the script, in other cases, performance is a bit worse, because of the ChatGPT translation directly in its context window to compare some data with predefined information that is in English. The recursive part of the script is executed until the valid information is found, it means that when information is not found the results variable contains URL to the possible webpages instead of real data, so when there are no URLs left in the results variable than functions stop and return a valid object to the main function. But while it is still running the GptAnalyser spend about 1-2 requests to the OpenAI ChatGPT API, due to different flow for receiving even **description** or **processes** information. Returned object that formatted into a more valid structure and returned as a response to the **Aggregation service** request.

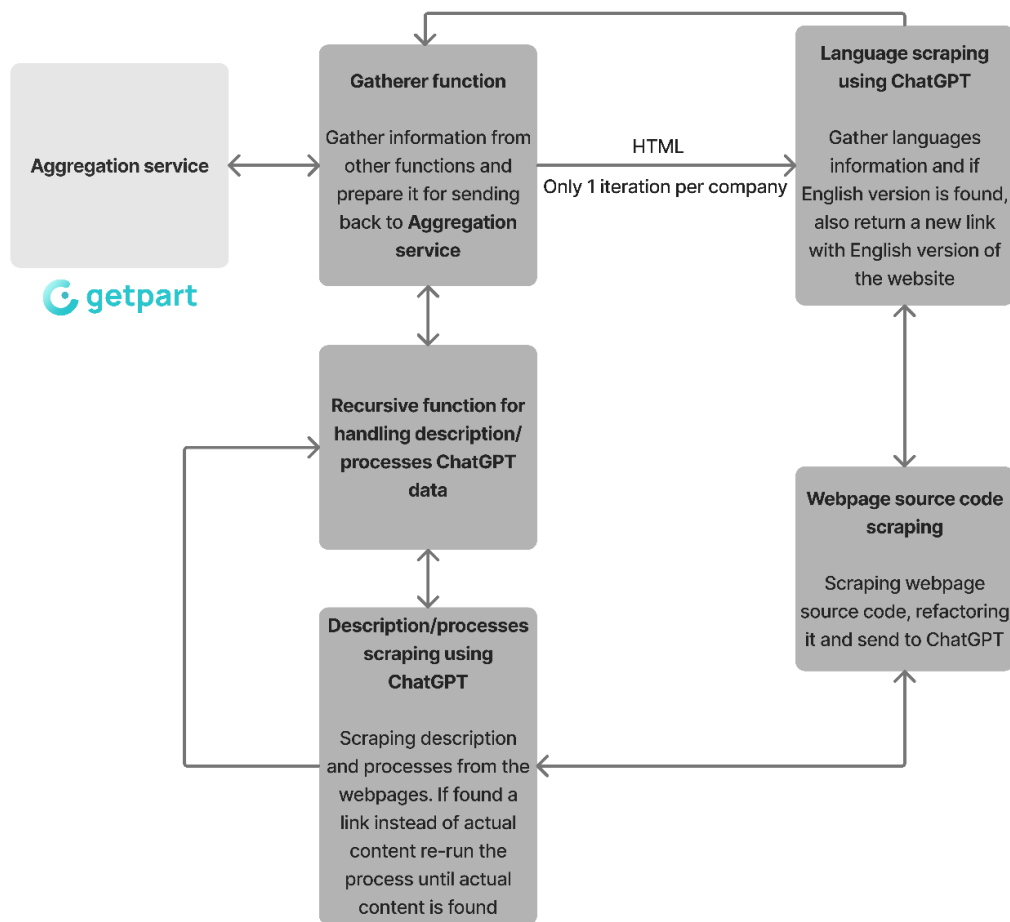


Figure 8. Simplified AI service structure

Figure 8 shows a simplified version of the AI services structure. Figure 8 describes the main parts of the AI service without going deep into the different functions calling.

6 Results

6.1 Results evaluation

The result is a prototype product, the development of which uses a variety of modern technologies in different areas of software development. Various pros and cons of using certain technologies were identified as the prototype was developed; some approaches had to be abandoned due to their inability to fulfill the assigned tasks or when better options were found for a specific type of task.

Firstly, a service was created whose purpose was to obtain structured data from the e-Business Register and its further processing. The main criteria for success were finding relevant information to fill out all the fields in the main application database. But since the e-Business Register does not provide complete information on this matter, the main goal was to identify and process only those companies that have their own, working website that is matched in the e-Business Register API output. Since there are a huge number of data processing companies and the data received from their websites is unstructured, it was decided to use AI for these purposes.

Since AI-based products became popular for a huge number of people just a couple of years ago, there are still not many publicly available options to use. The time-proven ChatGPT from OpenAI became the only one at the time of approval of the LLM thesis topic that was available to the general public to use and the most capable for requirements.

The use of this LLM made it possible to get rid of the problems of data shortage by obtaining unstructured data from company websites. Thus, this AI service helped achieve the main criterion for success, namely filling out all the data for the main application database.

6.1.1 Testing the prototype workflow

The prototype version of the application is only capable of handling companies from only one category – **Machining**. As was discovered, there are around 83 existing companies under this category in the e-Business Register. Since every ChatGPT API costs money,

there is a strictly limited budget for testing the prototype in this thesis. The prototype was tested on about 15-20 different companies from the **Machining** category in total. However, not all tested companies' websites are presented in the prototype workflow testing, because of their website data absence in the register, e.g. Hissmekano Estonia OÜ website is tested, but not in the prototype workflow test, and the results can be seen in the next chapter.

Due to ChatGPT API isn't free and the randomness of the processes in 20 random companies from the **Machining** category, for testing the prototype workflow selected 6 companies' websites which all had relevant required information. It is needed, because random, e.g. 20 companies parsed from the e-Business Register JSON file may not contain valid companies at all, due to the **Machining** category being too abstract. The criterion of success is to have 5-6 (there is 1 website with a language switch button bug) companies to be shown in the Getpart application website UI with a response with successes and rejected company's registry codes as was shown in Figure 5. Returning the incomplete data is not a problem in a thesis prototype, the customers who ordered this application product development allow data to be incomplete. Further development will most likely fix this issue.

The verification of returned from ChatGPT API data is done manually due to the small number of companies to process. The final decision about verification data is not yet decided. The most important data comes from the AI service, these data is processes, supported languages, and description. From these data, information about processes is the most sensitive, because it can be presented differently on different websites. Unlike description or language data, processes should be validated by providing ChatGPT with a more detailed processes list to cover all possible corner cases. Due to the huge number of companies' websites to process, to be able to test the current processes list on which ChatGPT API relies when searching for the information, it's mandatory to test a prototype on some certain list of companies' websites. If the output data is mostly valid, then the **verification** processes list is good for processing other companies' websites.

```

{
  "processed_companies": 6,
  "successful_ids": [
    11374695,
    14352866,
    14540036,
    14187442,
    11117403
  ],
  "rejected_ids": [
    10793740
  ]
}

```

Figure 9. Aggregation service output

As can be seen from Figure 7, the total amount of processed companies was 6, 5 of them were added to the Getpart database with a **Supplier** status and can be seen in Appendix 7 images from the Getpart front-end application. The website scraper script faced a language switch button bug and couldn't process this company but notified it in the **rejected_ids** field. Based on the manual verification, all parsed companies contained valid processes data. It is completed and without any **hallucinations** from ChatGPT.

6.1.2 Accuracy comparison evaluation

The accuracy problem of returned data from the AI Service is one of the most important, due to the randomness of responses from ChatGPT API it's quite difficult to create a service that returns relevant data constantly, or at least returns not artificially generated data. To be able to increase the accuracy of the ChatGPT responses a few tests with different prompts were tested to understand which prompt is better as an instruction for ChatGPT to compare processes information. As an example, the prompt that is presented in Appendix 6 was chosen to test ChatGPT on the websites of these companies: OÜ Istrek, Scandisteel OÜ, and Hissmekano Estonia OÜ. The non-highlighted version of the prompt in Appendix 6 does not have its representation in this thesis, however the non-highlighted version is the same but without using symbols like exclamation marks and sentences in upper case.

Table 3 shows the difference in responses of processes using the GPT-4 Turbo model of ChatGPT API. For every highlighted and non-highlighted test 5 attempts were allocated.

Table 3. Overview of different responses of processes data using GPT-4 Turbo

Company name	Using highlighted prompt	Using non-highlighted prompt	Valid processes
OÜ Istrek	3/5 of the responses contained: CNC Milling, CNC Turning, Stamping 2/5 of the responses contained: CNC Milling, CNC Turning, Stamping, Drilling/Tapping	5/5 of the responses contained: CNC Milling, CNC Turning, Stamping	CNC Milling, CNC Turning, Stamping, Drilling/Tapping or Drilling
Scandisteel OÜ	3/5 of the responses contained: Sheet metal Bending, MIG/MAG welding, TIG welding, MMA welding 2/5 of the responses contained: Sheet metal Bending, MIG/MAG welding, TIG welding	5/5 of the responses contained: Sheet metal Bending	Sheet metal Bending, MIG/MAG welding, TIG welding
Hissmekano Estonia OÜ	4/5 of the responses contained: CNC Milling, CNC Turning, Assembly 1/5 of the responses contained: CNC Milling, CNC Turning, Assembly, Sawing	4/5 of the responses contained: CNC Milling, CNC Turning, Assembly 1/5 of the responses contained: CNC Milling, CNC Turning, Assembly, Sawing	CNC Milling, CNC Turning, Assembly, Sawing

All 3 websites have relevant processes information, those sites were selected manually to make sure that they have needed data to test on. All these websites have different layouts, and information is presented differently, e.g. OÜ Istrek has processes information on the main page, while the Hissmekano Estonia OÜ has a separate page for those processes called **Services**. Unlike both of those websites, Scandisteel OÜ contains only generic names for those processes on the main page and in the header, to be able to get actual processes names, the application prototype should navigate to a specific page and find information there.

As can be seen from OÜ Istrek website, using a highlighted prompt it was possible to get more data, rather than using a non-highlighted prompt which returned 1 process less. However, only 40% of the responses from OÜ Istrek websites returned complete data that should be returned. The missing process is **Drilling/Tapping**, this processes doesn't exists on the website with this name, however, there is a processes called **Drilling**. Getpart has its list of processes that the current application prototype should rely on, and in this case, it contains **Drilling/Tapping**, not **Drilling**. In the context of this application, it's the same processes, so in 40% of the responses, ChatGPT understood those names while comparing them using a pre-defined prompt from Appendix 6. While using a non-highlighted prompt, it does not manage to understand the prompt properly, and in all 5 test runs returned all processes except **Drilling/Tapping**.

Scandisteel OÜ website has a more complex structure, processes aren't located in one place. All processes are allocated on different pages under the generic names of the processes, e.g. more precise names of welding processes: MIG/MAG welding and TIG welding, located on a welding page. More surprisingly here the non-highlighted prompt managed to figure only **Sheet metal Bending** process out, while the highlighted prompt found other **MIG/MAG welding**, **TIG welding**, **MMA welding** processes in addition to **Sheet metal Bending**. Highlighted prompt, however, has all 4 processes in 80% of test runs, while in 20% it has artificially generated **MMA welding**.

Hissmekano Estonia OÜ has the most primitive website, it has all the processes on a separate page called **Services**. In the case of Hissmekano Estonia OÜ the highlighted and non-highlighted results were absolutely the same, both of those prompts returned in 80% of the responses **CNC Milling**, **CNC Turning**, **Assembly** processes, missing only in 20% of one process named **Sawing**. More detailed information about returning incomplete data using Hissmekano Estonia OÜ as an example is already described in chapter 4.2.1.

In conclusion, there is plenty of work to increase the accuracy of the returned data and avoid a small percentage of artificially generated data, but it is already not in the scope of this thesis. As can be seen from Table 3, the more complex the website structure is, the greater the difference in returned data using different types of prompts is visible. The current situation with accuracy is acceptable for the prototype, but moving application product to the production level will require improving accuracy and avoiding AI-generated data as much as possible.

6.2 Feedback and the future of the project

The obtained solution covers all the requirements of the customer, the working prototype allows to expand the company's database of the main application and offers customers a better service. If at the beginning all the suppliers who wanted to be listed on the Getpart website had to register manually, now this process works differently. If the potential client wants to be listed on the Getpart website, most likely its company was already registered if the company website was presented in the e-Business Register company's list, if so then the client requests registration, and after manual verification from the administration client becomes a member of the company.

And there is huge potential for the further development of this project. From code optimization, switching to a newer version of ChatGPT, or even using self-trained LLM to adapting this service work not only with Estonian registered companies but also with other EU business registers.

Other EU business registers might also require the development of something new for this service or even restructuring to be able to support their standards.

The possibility to make ChatGPT model text generation more effective and precise using fine-tuning provided by OpenAI. This approach could help to avoid repetitively adding prompts, instead, it's possible to "pre-train" a model by uploading a data set of examples. It could make a model more precise for a specific requirement of the project.

Providing AJAX support to the service also brings additional companies' data. Even though websites by specified categories are primitive, there could be a small number of those that are using AJAX for dynamic content loading. Currently, the service supports webpages with dynamic content loading and service-side rendering by using Chromium browser under the hood, but out of the scope falls AJAX-based websites due to their complicated tracking with a huge number of input sites.

Adding either weekly or monthly support to re-request the company's data from the e-Business Register to find newly registered companies that can be relevant to the Getpart application.

7 Summary

The goal of this thesis was to analyze the possibility and potential of using LLM as an innovative method for solving problems like unstructured data scraping based on collecting and aggregating corporate profile data from a company's website that is publicly available from the e-Business Register of Estonia.

For that purpose, decided to build a prototype for collecting the company's data firstly from the e-Business Register, then sending a request based on the existence of the website in the provided data set from the register, building a separate service for using OpenAI ChatGPT to retrieve relevant data from the company website and aggregate this data into the Getpart application back-end.

In the thesis analysis, some limitations were identified in the work of different aspects of existing libraries due to the high number of input data. In that case, several different variants of libraries were tested to handle such huge inputs. So, the best outcome was achieved after the detailed investigation at this moment.

References

- [1] Cloudflare, “What is a large language model (LLM)?,” [Online]. Available: <https://www.cloudflare.com/learning/ai/what-is-large-language-model>. [Accessed 16 February 2024].
- [2] A. Mirzaei, “LLMs Can Convert Unstructured Data into Structured Gold,” september 2023. [Online]. Available: <https://www.linkedin.com/pulse/llms-can-convert-unstructured-data-structured-gold-ali-mirzaei>. [Accessed 16 February 2024].
- [3] J. von Brocker, A. Hevner, A. Maedche, “Introduction to Design Science Research,” september 2020.
- [4] D. Breeuwer, “AI Web Scraper,” [Online]. Available: <https://github.com/dirkjbreeuwer/gpt-automated-web-scraper>. [Accessed 22 February 2024].
- [5] S. Vaniukov, “NLP vs LLM: A Comprehensive Guide to Understanding Key Differences,” february 2024. [Online]. Available: <https://medium.com/@vaniukov.s/nlp-vs-llm-a-comprehensive-guide-to-understanding-key-differences-0358f6571910>. [Accessed 26 February 2024].
- [6] GrowthLoop, “Large language model (LLM),” february 2024. [Online]. Available: <https://www.growthloop.com/university/article/llm>. [Accessed 1 March 2024].
- [7] M. Stevens, “Best GPU for LLM Inference and Training,” march 2024. [Online]. Available: <https://bizon-tech.com/blog/best-gpu-llm-training-inference#:~:text=Small%20to%20medium%20models%20can,recommended%20for%20training%20and%20inference>. [Accessed 3 March 2024].
- [8] M. Nuñez, “LLaMA 2: How to access and use Meta’s versatile open-source chatbot right now,” july 2023. [Online]. Available: <https://venturebeat.com/ai/llama-2-how-to-access-and-use-metas-versatile-open-source-chatbot-right-now>. [Accessed 5 March 2024].
- [9] S. Gholami, M. Omar, “Do Generative Large Language Models need billions of parameters?,” september 2023.
- [10] B. Marie, “Run Llama 2 70B on Your GPU with ExLlamaV2,” september 2023. [Online]. Available: <https://kaitechup.substack.com/p/run-llama-2-70b-on-your-gpu-with>. [Accessed 8 March].
- [11] Ainaive, “The AI Showdown: OpenAI’s GPT vs Google’s Gemini vs Meta’s LLaMA 2,” february 2024. [Online]. Available: <https://www.linkedin.com/pulse/ai-showdown-openais-gpt-vs-googles-gemini-metas-llama-2-ainavehq-pfylc>. [Accessed 10 March].
- [12] H. Ye, T. Liu, A. Zhang, W. Hua, W. Jia, “Cognitive Mirage: A Review of Hallucinations in Large Language Models,” september 2023.
- [13] Epam, “top 5 NodeJS pros and cons: what they mean for your project,” april 2024. [Online]. Available: <https://anywhere.epam.com/en/blog/node-js-pros-and-cons>. [Accessed 3 April].
- [14] M. Heller, “What is Node.js? The JavaScript runtime explained,” april 2020.
- [15] Microservices.io, “Microservice Architecture pattern,” [Online]. Available: <https://microservices.io/patterns/microservices.html>. [Accessed 5 April].

- [16] Expressjs, “Fast, unopinionated, minimalist web framework for Node.js,” [Online]. Available: <https://expressjs.com>. [Accessed 6 April].
- [17] A. Soni, V. Ranga, “API Features Individualizing of Web Services: REST and SOAP,” july 2019.
- [18] M. Galle’, “Investigating the Effectiveness of BPE: The Power of Shorter Sequences,”
- [19] M. Fromm, “Design of LLM prompts for iterative data exploration,” 2024
- [20] O. Brown, ““Hallucinating” AIs Sound Creative, but Let’s Not Celebrate Being Wrong,” october 2023. [Online]. Available: <https://thereader.mitpress.mit.edu/hallucinating-ais-sound-creative-but-lets-not-celebrate-being-wrong>. [Accessed 12 April].
- [21] NonStop io Technologies, “What is Prompt Engineering? — Part 2: Properties of Effective Prompts,” august 2023. [Online]. Available: <https://www.linkedin.com/pulse/what-prompt-engineering-part-2-properties-effective-prompts>. [Accessed 13 April].
- [22] Gitlab, “What are the benefits of a microservices architecture?,” september 2022. [Online]. Available: <https://about.gitlab.com/blog/2022/09/29/what-are-the-benefits-of-a-microservices-architecture>. [Accessed 13 April].
- [23] A. Ranjan, A. Sinha, R. Battewad, “JavaScript for Modern Web Development,” [Online]. Available: https://books.google.ee/books?hl=ru&lr=&id=b2bdDwAAQBAJ&oi=fnd&pg=PT24&dq=The+JavaScript+flexibility&ots=6gcC-CCrIW&sig=qxDYUXHD2GbLUuR-Homwq94ahiw&redir_esc=y#v=onepage&q=The%20JavaScript%20flexibility&f=false. [Accessed 14 Aprill].
- [24] Centre of Registers and Information Systems, “Determining the main field of activity,” april 2024. [Online]. Available: <https://www.eesti.ee/en/doing-business/establishing-a-company/determining-the-main-field-of-activity>. [Accessed 13 April].
- [25] S. Raschka, J. Patterson, C. Nolet, “Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence,” 2020 .[Online]. Available: <https://www.mdpi.com/2078-2489/11/4/193>. [Accessed 5 May].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis¹

I Aleksandr Lerko

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis, "Exploring a Novel Approach for Aggregation of Company Profile Data" supervised by Pavel Tšikul
 - 1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
 - 1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

12.05.2024

¹ The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Final result of description prompt

```
f""  
    !!!First, you should analyze it and help me to find company description  
and make a summary  
    of it. !!!If there is no description on current page found, you should  
find a link to a page where it can be described  
    , most likely it will be about us page!!!, but (DO IT ONLY IF THIS LINK  
TO A POSSIBLE SERVICES PAGE URL IS NOT THE  
    SAME AS THIS - {url}).  
  
    The link will be either in <a> or <link> tags, but also  
    !!!PAY ATTENTION THAT LINK CAN BE EITHER A FULL NEW URL TO THE NEXT PAGE  
OR AN ADDITIONAL PREFIX. IF IT'S ADDITIONAL  
    PREFIX, THEN COMBINE IT WITH THIS URL - {url}.  
  
    If you found only link to another page, then return it to summary field  
  
    !!!ALL THIS INFORMATION MIGHT BE LANGUAGE SENSITIVE, SO ALSO PAY  
ATTENTION ON IT!!!  
    ""
```

Appendix 3 – Differences between GPT-4 Turbo and GPT-3.5

Turbo company description responses

GPT-4 Turbo:

"description": "HME was established in 2007 with an aim to offer high quality complex services in the field of mechanics: from the customer's idea to our realisation. In the first year of activity, they launched manufacturing in a 940 m2 site and started to manufacture different stainless steel pump details. Additionally, they started preparing the transition of the production of different elevator details from Sweden to Estonia, which was completed in 2008 etc..."

GPT-3.5 Turbo:

"description": "The company description can be found on the 'About Us' page. Here is the link to the 'About Us' page: <https://www.hme.ee/about-us>"

Appendix 4 – List of processes provided by Getpart

```
possible_processes = [  
    'Assembly',  
    'Casting',  
    'Plasma cutting',  
    'Laser cutting',  
    'Water jet',  
    'Profile cutting',  
    'Extrusion',  
    'Forging',  
    'Sheet metal Bending',  
    'Sheet metal Rolling',  
    'Profile bending',  
    'CNC Milling',  
    'CNC Turning',  
    'Drilling/Tapping',  
    'Punching',  
    'Sawing',  
    'Shearing',  
    'Stamping',  
    'TIG welding',  
    'MIG/MAG welding',  
    'MMA welding',  
    'Flux Core Arc Welding',  
    'Laser beam welding',  
    'Spot welding',  
    'Folding'  
]
```

Appendix 5 – Prompts for retrieving processes

With highlighting:

```
get_services_prompt = f"""
```

```
    !!!Second, identify information about the services.
```

```
    If it's possible to find a link to the services either page or section,  
    !!!THEN PRIORITIZE THIS COMPLETED LINK OVER RETURNING
```

```
    ACTUAL SERVICES!!!, but (DO IT ONLY IF THIS LINK TO A POSSIBLE SERVICES  
    PAGE URL IS NOT THE SAME AS THIS - {url}).
```

```
    Most likely this info will be either in <a> or <link> tags, but also
```

```
    !!!PAY ATTENTION THAT LINK CAN BE EITHER A FULL NEW URL TO THE NEXT PAGE  
    OR AN ADDITIONAL PREFIX. IF IT'S ADDITIONAL
```

```
    PREFIX, THEN COMBINE IT WITH THIS URL - {url}.
```

```
    If you are on services page, then find a section where services are  
    described and then return of all services
```

```
    as list of strings.
```

```
    """
```

Without highlighting:

```
get_services_prompt = f"""
```

```
    Second, identify information about the services.
```

```
    If it's possible to find a link to the services either page or section,  
    then prioritize this completed link over
```

```
    returning actual services, but (do it only if this link to a possible  
    services page url is not the same as this - {url}).
```

```
    Most likely this info will be either in <a> or <link> tags, but also
```

```
    pay attention that link can be either a full new url to the next page or  
    an additional prefix.
```

```
    If it's additional prefix, then combine it with this url - {url}.
```

```
    If you are on services page, then find a section where services are  
    described and then return of all services
```

```
    as list of strings.
```

```
    """
```

Appendix 6 – Processes comparison prompt

```
return f"""
        Here is the list of services on !!!!! WHICH YOU SHOULD RELY
ON. COMPARE FOUNDED BY YOU SERVICES ONE BY
        ONE WITH THOSE IN THE PROVIDED FOR YOU LIST. IF THERE IS NO
SIMILAR SERVICES THEN DO NOT WRITE ANYTHING
        RETURN EMPTY LIST!!!!.

        {possible_processes}










        !!!PAY ATTENTION THAT SERVICES NAMES COULD BE IN DIFFERENT
LANGUAGES, IF LANGUAGE OF THE WEB PAGE IS NOT
        ENGLISH!!!
        If so try to manually translate them and compare. Returned
names of services should be strings with only
        it's names

        If you found a service that is similar to a services in above
list, then return name of this service
        from a list.

        OUTPUT SHOULD BE A LIST OF STRINGS (WITH SERVICES NAMES)

        returned object is json with filed "services": []
"""
```

Appendix 7 – Test results appeared on Getpart website

	Limitless Machinery OÜ Estonia, Harju County, 10149, Tallinn, Saani tn 2/2-23 CNC Milling	Request price View details Edit 
	BRASS MACHINING OÜ Estonia, Põlva County, 63710, Põlva vald, Tehase tn 6 CNC Milling CNC Turning Sawing	Request price View details Edit 
	Scandisteel OÜ Estonia, Harju County, 74207, Jõelähtme vald, Käpa tee 2 Sheet metal Bending TIG welding MIG/MAG welding	Request price View details Edit 
	Irontec OÜ Estonia, Tartu County, 50412, Tartu linn, Ilmatsalu tn 3 Assembly CNC Milling CNC Turning Sawing TIG welding +1	Request price View details Edit 
	Radius Machining OÜ Estonia, Harju County, 75312, Rae vald, Sära tee 8 Assembly CNC Milling CNC Turning	Request price View details Edit 