SEMANTIC DATA ANALYSIS

Major project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

In

Computer Science and Engineering

By

TEJASWEE TEWARI(181362)

UNDER THE SUPERVISION OF

Dr. Ruchi Verma

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

CERTIFICATE

This is to certify that the work which is being presented in the project report titled ***“Semantic Data Analysis”*** is in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by ***“Tejaswee Tewari(181362)”*** during the period from January 2022 to May 2022 under the supervision of ***Dr. Ruchi Verma***, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Tejaswee Tewari (181362)

The above statement is correct to the best of my knowledge.

Dr. Ruchi Verma

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to Almighty God for his divine blessing in making it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor ***Dr. Ruchi Verma*** Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of ***"Data Science"*** to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to ***Dr. Ruchi Verma****,* Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Student Name

Tejaswee Tewari (181362)

Table of Content

List of Figures

	management systems	

Abstract

The problem we would be dealing with is the inconsistency in the Pharmaceutical drug data. All the data available to us through the internet, lacks structure and is very ambiguous. The objective here is to analyze this data and meaningfully co-relate them. And with the help of this co-related data we can structure it and derive hidden relationships.

To achieve this, we need to build a fuzzy search engine wherein we can merge data from various sources and derive hidden relationships

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

We as humans have a tendency to constantly learn and co-relate the information that we already know. We are surrounded by data, the internet, web pages, research papers, articles, blogs, etc. The current worldwide web is document-driven, where-in different web pages have independent data associated, which are further linked together.
All of this data that is available lacks structure and is very ambiguous. The aim of this project is to semantically analyze the data in the pharmaceutical domain and co-relate them with the help of ontologies. With the help of this correlated data, we can structure it according to our needs and derive hidden relationships.

1.2 Problem Statement

The problem statement for this project is: **“Solving the problems of inconsistency in the Pharmaceutical drug data”**.
After noticing the inconsistency of the data of clinical trials available, it became a difficult task to correlate them with other datasets, moreover, the regulations and restrictions on the raw ingredients and the salts used for making new drugs are different in different countries.
Making a search engine for this problem will not only help us to determine the uniqueness of the drug but also aid in solving multiple problems in a generalized way. Paracetamol example-different regulations in different countries.

1.3 Objective

The objective here is to build a fuzzy search engine for the pharmaceutical domain wherein we can merge data from various sources and derive hidden relationships using the power of ontologies and run a query on them to extract information using the search engine.

1.4 Methodology

The methodology behind this project is a Semantic Analysis of the data. It is a process of identifying the exact meaning and tone behind the unstructured text, herein we will scrape the drug-related data from various sources and annotate it to derive the semantics of the data and construct ontologies out of them, which will further help us in running the queries on the extracted data using the fuzzy search engine.

CHAPTER 2

REVIEW OF LITERATURE

2.1 Survey:

To deduce the topic of our study we followed a set of steps that helped us to scrape the web and extract out the potential keyword that would help us in identifying the problems in this domain. The set of steps are mentioned as follows:

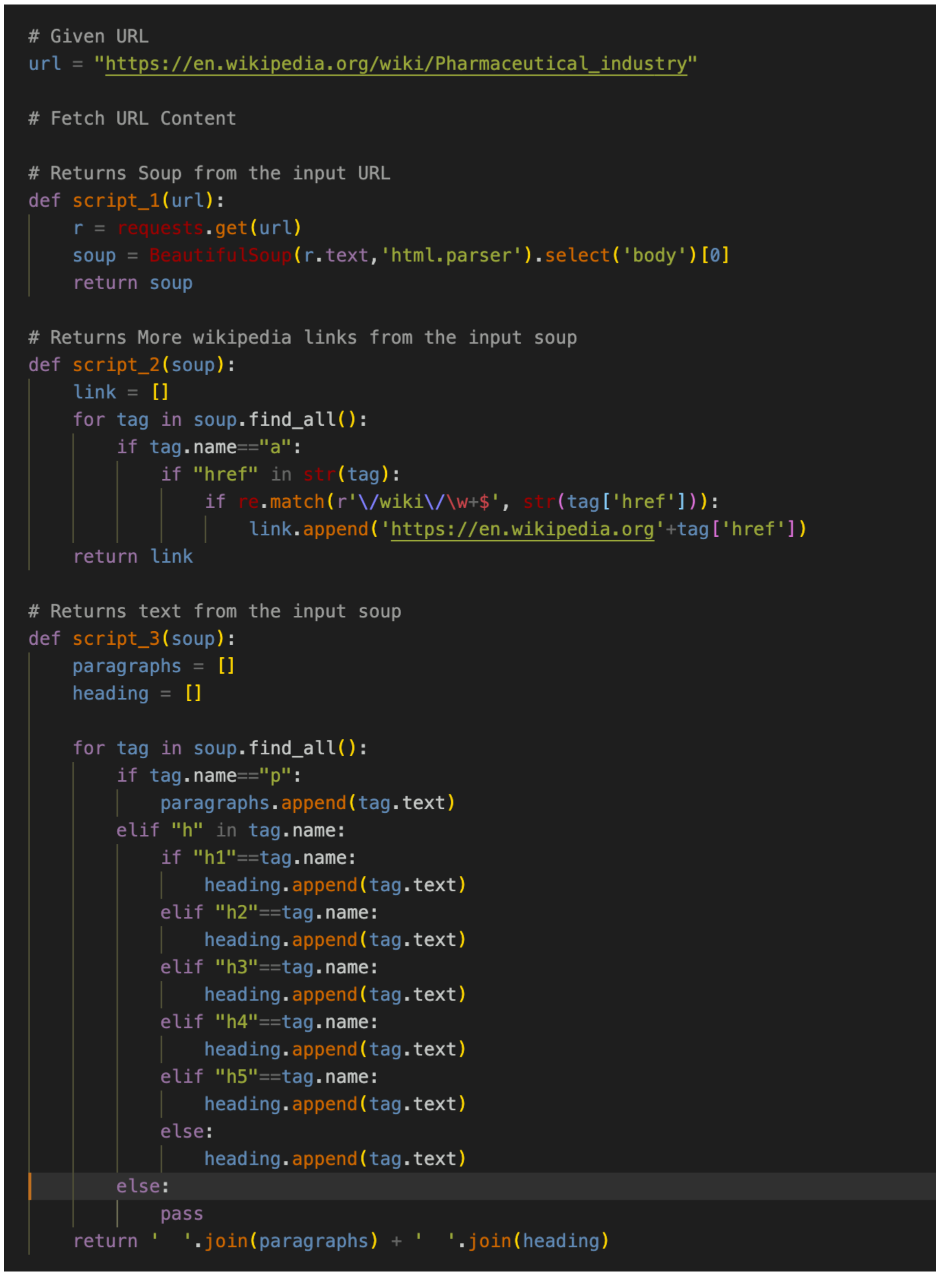
2.1.1 Creation of Scripts:

- **Script 1:**
  This script takes input as the link of the webpage(Wikipedia) that was fed to the script i.e. a Wikipedia page that is inlined with the pharmaceutical domain and returns the HTML content on the page.

- **Script 2:**
  This script takes the Script 1 output and collects all the Wikipedia links from the HTML data and then feeds it again to the Script 1. This process then keeps on repeating in a loop.

- **Script 3:**
  This script takes the Script 1 returned HTML data as input and parses and annotates the data to extract the potential keywords that are of interest using the spacy library and then further arranges those keywords in descending order of the number of times it was mentioned.

Once we got the list of the keywords in descending order of their occurrence, we manually selected the ones that seemed to be relevant to the field of our study, omitting the remaining.

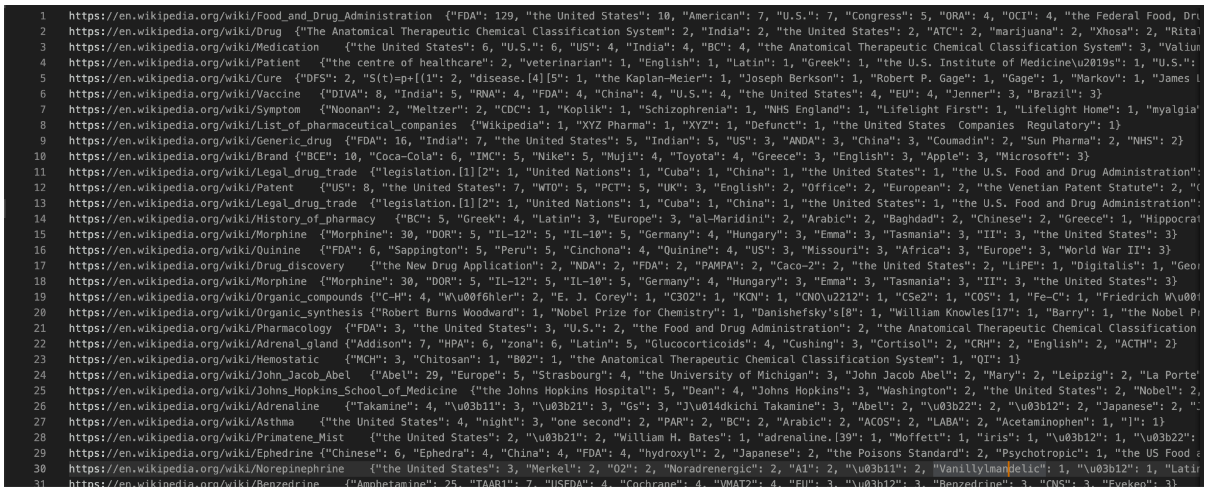


```python
# Given URL
url = "https://en.wikipedia.org/wiki/Pharmaceutical_industry"

# Fetch URL Content

# Returns Soup from the input URL
def script_1(url):
    r = requests.get(url)
    soup = BeautifulSoup(r.text,'html.parser').select('body')[0]
    return soup

# Returns More wikipedia links from the input soup
def script_2(soup):
    link = []
    for tag in soup.find_all():
        if tag.name=="a":
            if "href" in str(tag):
                if re.match(r'\/wiki\/\w+$', str(tag['href'])):
                    link.append('https://en.wikipedia.org'+tag['href'])
    return link

# Returns text from the input soup
def script_3(soup):
    paragraphs = []
    heading = []

    for tag in soup.find_all():
        if tag.name=="p":
            paragraphs.append(tag.text)
        elif "h" in tag.name:
            if "h1"==tag.name:
                heading.append(tag.text)
            elif "h2"==tag.name:
                heading.append(tag.text)
            elif "h3"==tag.name:
                heading.append(tag.text)
            elif "h4"==tag.name:
                heading.append(tag.text)
            elif "h5"==tag.name:
                heading.append(tag.text)
            else:
                heading.append(tag.text)
        else:
            pass
    return ' '.join(paragraphs) + ' '.join(heading)
```

Fig 2.1 Code of the scripts used for parsing and scraping

2.1.2 Analysing the Script output:

Once we were done with this process and had a considerable amount of data, we manually went through the various keywords and associated data with it, prioritizing the ones we found comparatively more relevant like ‘FDI’ had shown up a number of times, ‘generic drug’ was also another popular keyword in the list repeating multiple time. So, for our ease, we made a list of those relevant keywords manually for future use.



```
https://en.wikipedia.org/wiki/Food_and_Drug_Administration  {"FDA": 129, "the United States": 10, "American": 7, "U.S.": 7, "Congress": 5, "ORA": 4, "OCI": 4, "the Federal Food, Dru
https://en.wikipedia.org/wiki/Drug  {"The Anatomical Therapeutic Chemical Classification System": 2, "India": 2, "the United States": 2, "ATC": 2, "marijuana": 2, "Xhosa": 2, "Rital
https://en.wikipedia.org/wiki/Medication    {"the United States": 6, "U.S.": 6, "US": 4, "India": 4, "BC": 4, "the Anatomical Therapeutic Chemical Classification System": 3, "Valiun
https://en.wikipedia.org/wiki/Patient   {"the centre of healthcare": 2, "veterinarian": 1, "English": 1, "Latin": 1, "Greek": 1, "the U.S. Institute of Medicine\u2019s": 1, "U.S.":
https://en.wikipedia.org/wiki/Cure  {"DFS": 2, "S(t)=p+[(1": 2, "disease.[4][5": 1, "the Kaplan-Meier": 1, "Joseph Berkson": 1, "Robert P. Gage": 1, "Gage": 1, "Markov": 1, "James L
https://en.wikipedia.org/wiki/Vaccine   {"DIVA": 8, "India": 5, "RNA": 4, "FDA": 4, "China": 4, "U.S.": 4, "the United States": 4, "EU": 4, "Jenner": 3, "Brazil": 3}
https://en.wikipedia.org/wiki/Symptom   {"Noonan": 2, "Meltzer": 2, "CDC": 1, "Koplik": 1, "Schizophrenia": 1, "NHS England": 1, "Lifelight First": 1, "Lifelight Home": 1, "myalgia'
https://en.wikipedia.org/wiki/List_of_pharmaceutical_companies  {"Wikipedia": 1, "XYZ Pharma": 1, "XYZ": 1, "Defunct": 1, "the United States  Companies  Regulatory": 1}
https://en.wikipedia.org/wiki/Generic_drug  {"FDA": 16, "India": 7, "the United States": 5, "Indian": 5, "US": 3, "ANDA": 3, "China": 3, "Coumadin": 2, "Sun Pharma": 2, "NHS": 2}
https://en.wikipedia.org/wiki/Brand {"BCE": 10, "Coca-Cola": 6, "IMC": 5, "Nike": 5, "Muji": 4, "Toyota": 4, "Greece": 3, "English": 3, "Apple": 3, "Microsoft": 3}
https://en.wikipedia.org/wiki/Legal_drug_trade  {"legislation.[1][2": 1, "United Nations": 1, "Cuba": 1, "China": 1, "the United States": 1, "the U.S. Food and Drug Administration":
https://en.wikipedia.org/wiki/Patent    {"US": 8, "the United States": 7, "WTO": 5, "PCT": 5, "UK": 3, "English": 2, "Office": 2, "European": 2, "the Venetian Patent Statute": 2, "(
https://en.wikipedia.org/wiki/Legal_drug_trade  {"legislation.[1][2": 1, "United Nations": 1, "Cuba": 1, "China": 1, "the United States": 1, "the U.S. Food and Drug Administration":
https://en.wikipedia.org/wiki/History_of_pharmacy   {"BC": 5, "Greek": 4, "Latin": 3, "Europe": 3, "al-Maridini": 2, "Arabic": 2, "Baghdad": 2, "Chinese": 2, "Greece": 1, "Hippocrat
https://en.wikipedia.org/wiki/Morphine  {"Morphine": 30, "DOR": 5, "IL-12": 5, "IL-10": 5, "Germany": 4, "Hungary": 3, "Emma": 3, "Tasmania": 3, "II": 3, "the United States": 3}
https://en.wikipedia.org/wiki/Quinine   {"FDA": 6, "Sappington": 5, "Peru": 5, "Cinchona": 4, "Quinine": 4, "US": 3, "Missouri": 3, "Africa": 3, "Europe": 3, "World War II": 3}
https://en.wikipedia.org/wiki/Drug_discovery    {"the New Drug Application": 2, "NDA": 2, "FDA": 2, "PAMPA": 2, "Caco-2": 2, "the United States": 2, "LiPE": 1, "Digitalis": 1, "Geor
https://en.wikipedia.org/wiki/Morphine  {"Morphine": 30, "DOR": 5, "IL-12": 5, "IL-10": 5, "Germany": 4, "Hungary": 3, "Emma": 3, "Tasmania": 3, "II": 3, "the United States": 3}
https://en.wikipedia.org/wiki/Organic_compounds {"C-H": 4, "W\u00f6hler": 2, "E. J. Corey": 1, "C3O2": 1, "KCN": 1, "CNO\u2212": 1, "CSe2": 1, "COS": 1, "Fe-C": 1, "Friedrich W\u00f
https://en.wikipedia.org/wiki/Organic_synthesis {"Robert Burns Woodward": 1, "Nobel Prize for Chemistry": 1, "Danishefsky's[8": 1, "William Knowles[17": 1, "Barry": 1, "the Nobel Pr
https://en.wikipedia.org/wiki/Pharmacology  {"FDA": 3, "the United States": 3, "U.S.": 2, "the Food and Drug Administration": 2, "the Anatomical Therapeutic Chemical Classification
https://en.wikipedia.org/wiki/Adrenal_gland {"Addison": 7, "HPA": 6, "zona": 6, "Latin": 5, "Glucocorticoids": 4, "Cushing": 3, "Cortisol": 2, "CRH": 2, "English": 2, "ACTH": 2}
https://en.wikipedia.org/wiki/Hemostatic    {"MCH": 3, "Chitosan": 1, "B02": 1, "the Anatomical Therapeutic Chemical Classification System": 1, "QI": 1}
https://en.wikipedia.org/wiki/John_Jacob_Abel   {"Abel": 29, "Europe": 5, "Strasbourg": 4, "the University of Michigan": 3, "John Jacob Abel": 2, "Mary": 2, "Leipzig": 2, "La Porte'
https://en.wikipedia.org/wiki/Johns_Hopkins_School_of_Medicine  {"the Johns Hopkins Hospital": 5, "Dean": 4, "Johns Hopkins": 3, "Washington": 2, "the United States": 2, "Nobel": 2,
https://en.wikipedia.org/wiki/Adrenaline    {"Takamine": 4, "\u03b11": 3, "\u03b21": 3, "Gs": 3, "J\u014dkichi Takamine": 3, "Abel": 2, "\u03b22": 2, "\u03b12": 2, "Japanese": 2, "J
https://en.wikipedia.org/wiki/Asthma    {"the United States": 4, "night": 3, "one second": 2, "PAR": 2, "BC": 2, "Arabic": 2, "ACOS": 2, "LABA": 2, "Acetaminophen": 1, "]": 1}
https://en.wikipedia.org/wiki/Primatene_Mist    {"the United States": 2, "\u03b21": 2, "William H. Bates": 1, "adrenaline.[39": 1, "Moffett": 1, "iris": 1, "\u03b12": 1, "\u03b22":
https://en.wikipedia.org/wiki/Ephedrine {"Chinese": 6, "Ephedra": 4, "China": 4, "FDA": 4, "hydroxyl": 2, "Japanese": 2, "the Poisons Standard": 2, "Psychotropic": 1, "the US Food a
https://en.wikipedia.org/wiki/Norepinephrine    {"the United States": 3, "Merkel": 2, "O2": 2, "Noradrenergic": 2, "A1": 2, "\u03b11": 2, "Vanillylmandelic": 1, "\u03b12": 1, "Latin
https://en.wikipedia.org/wiki/Benzedrine    {"Amphetamine": 25, "TAAR1": 7, "USFDA": 4, "Cochrane": 4, "VMAT2": 4, "EU": 3, "\u03b12": 3, "Benzedrine": 3, "CNS": 3, "Evekeo": 3}
```

Fig 2.2 Glimpse of the output received after scraping and annotation.

2.1.3 Finding resources on the web:

After in-depth analysis and initial manual shortlisting of the keywords received in the output file, we manually searched the terms on the web to find some potential results with the prime focus on finding the problems associated with the search terms. To keep the track of the progress done, we used excel sheets to document our findings.

2.1.4 Identifying the problems:

Based on the script output, we searched the web for the problem areas connected to those keywords, while simultaneously asking the question of ourselves how data can solve such a problem, which made it easier for us to find the problem areas where we can derive a data-driven solution.

2.1.5 Identifying the resources of those problems:

Once the problem statements were derived, we searched the web for various databases consisting of all the features associated with the problem statement. In most of the cases, we had to club two or more datasets to get all the required features. We paid special attention to finding the resources for the problem statement because for a data science project, no matter how good the idea is, having the data for the problem statement is very crucial.

2.1.6 Narrowing down to one problem:

Based on the available data and resources we formulated 14 problem statements which are mentioned below:

1. I wish to manufacture a certain drug and export it to different countries. How can I keep a check on regulatory information?
2. Circulation of Fake/False drugs in the market.
3. A Gap in the communication between the drug regulatory bodies both at the center and state level in India.
4. Poor quality of generic drugs in the market in India.
5. Drug Shortage and increasing use of expensive and highly regulated medication.
6. Identify and target the biggest drivers of drug use.
7. Cases Registered Against E-Pharmacy Websites

8. Inter-State Sale of Drugs
9. Selling Scheduled Drugs
10. Poor Pharmacy Practice in India and other Asian countries.
11. Public Awareness regarding consumption of generic drugs
12. Solving the problems of inconsistency in the Pharmaceutical drug data.

Considering the availability of the data and the feasibility of solving the problem statement we narrowed it down to one problem i.e. - **Solving the problems of inconsistency in the Pharmaceutical drug data?**

CHAPTER 3

SYSTEM DEVELOPMENT

3.1 Workflow:

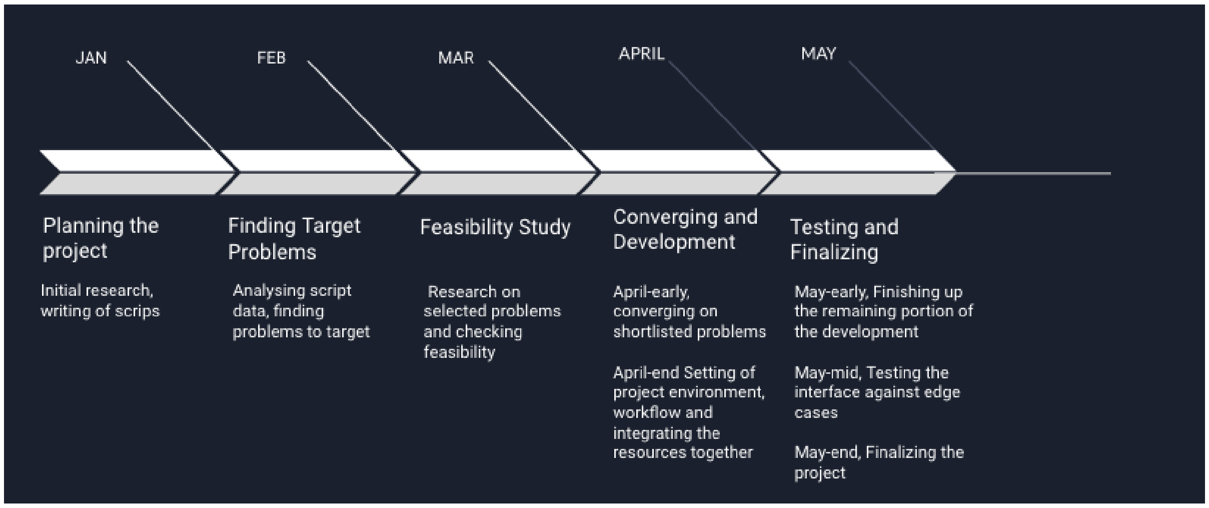

JAN
FEB
MAR
APRIL
MAY
Planning the project
Initial research, writing of scrips
Finding Target Problems
Analysing script data, finding problems to target
Feasibility Study
Research on selected problems and checking feasibility
Converging and Development
April-early, converging on shortlisted problems
April-end Setting of project environment, workflow and integrating the resources together
Testing and Finalizing
May-early, Finishing up the remaining portion of the development
May-mid, Testing the interface against edge cases
May-end, Finalizing the project

Fig 3.1 Work pipeline of project development

3.2 Architecture

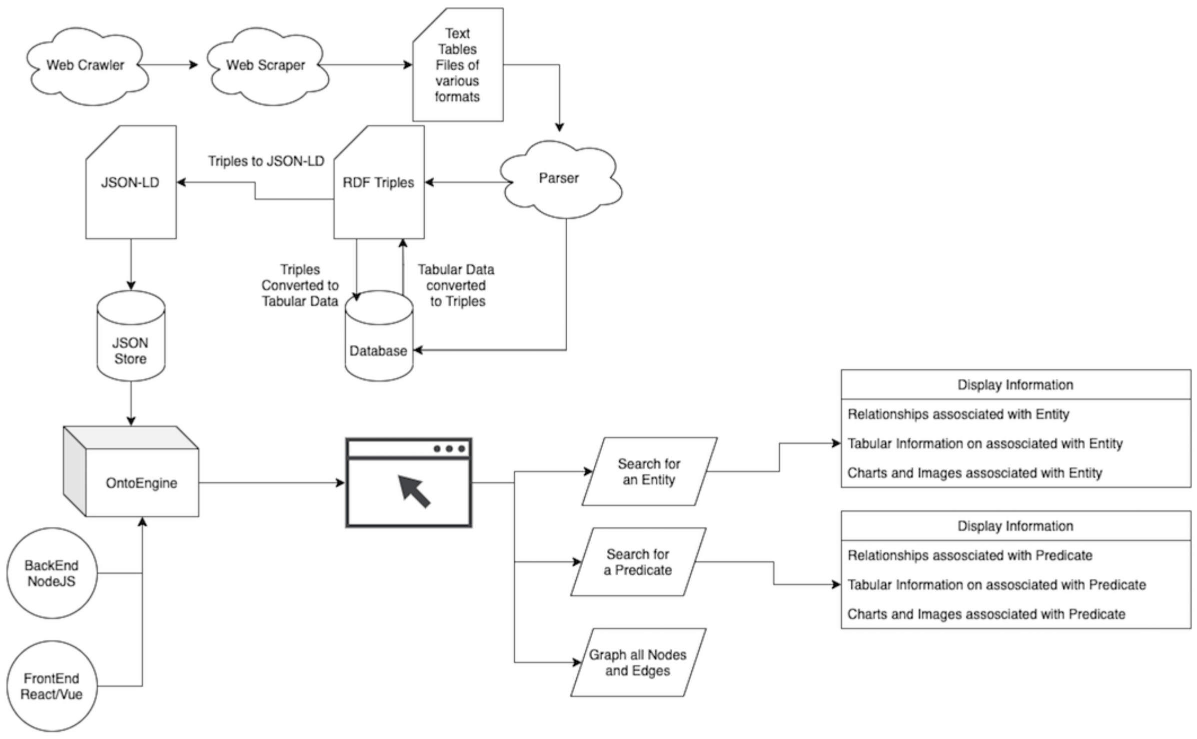

Web Crawler
Web Scraper
Text Tables Files of various formats
Parser
RDF Triples
Triples to JSON-LD
JSON-LD
Triples Converted to Tabular Data
Tabular Data converted to Triples
Database
JSON Store
OntoEngine
BackEnd NodeJS
FrontEnd React/Vue
Search for an Entity
Search for a Predicate
Graph all Nodes and Edges
Display Information
Relationships assosciated with Entity
Tabular Information on assosciated with Entity
Charts and Images assosciated with Entity
Display Information
Relationships assosciated with Predicate
Tabular Information on assosciated with Predicate
Charts and Images assosciated with Predicate

Fig 3.2 Architecture of the project

The above architecture can be divided into 3 sections:

- Parsing
- OntoEngine
- Database

3.3 Languages Used:

We have used a mix of languages to suit our needs, we have mainly used python for data-related work whereas Java Script for making a presentable working software highlighting our idea.

3.3.1 Python:

- Python is a quite popular language
- Guido van Rossum is the creator.
- Python can be used for a variety of things like
    - ML (Machine Learning)
    - For Server-side improvements.
    - Web-dev.
    - AI (Artificial intelligence).
    - Framework scripting

3.3.1 (a) What would python be able to do?

We primarily used python for the following tasks:

- To scrape information from the web using the beautiful soup library and many more.
- To annotate the data
- For extracting the keywords from the text we scraped from the web using the spacy library.

Furthermore, Python can also be used for:

- It can be used to make web-apps on the server

● It can be used to solve and perform complex arithmetic operations

● Python enables the user to associate it to various database frameworks out of the box.

● Python proves to be popular and very helpful in handling enormous amounts of data and information like DS (Data Science), Big Data, etc.

(b) Why Python?

We are using python language names because of the above-listed points as well as it is supported by the majority of platforms like Windows, Mac, Linux, etc. It is fairly straightforward. Python can be used when dealing with large amounts of information, moreover, it is an article arranged language.

3.3.2 Javascript

- Javascript is a famous text-based programming language.
- We can use javascript for
    - Making web pages interactive
    - Can be used on the client-side as well as the server-side
    - It gives web pages an interactive element that is highly beneficial in engaging users.

3.4 Libraries Used:

Libraries used in the project are as follows.

Dask: This is an open-source library that is mainly used for parallel computation. This was created by Mr. Matthew Rocklin, This library is a project which is maintained by the developers and organizations themselves. This library is made up of two components which are,

- Dynamic Task Scheduling
- Big Data Collection

Numpy: The library NumPy is a basic package for carrying out mathematical functions in Python Language. It is a library for python users that gives us multidimensional array objects. It helps us in a way that we can perform operations on arrays very quickly, and much more can also be done using the Numpy library.

Pandas: The library Pandas is a library that is written to be used in the python language for data analysis and data manipulation. Specifically, the Pandas library helps us in operations for manipulating the number tables and time series. The library Pandas can also help us in data analysis.

Fuse.js: This is a lightweight yet very powerful library that provides the facility of fuzzy-search. The best part is this has 0 inter-dependencies. Fuzzy searching is a technique consisting of various algorithms like Soundex, cosine, etc which are used to find strings that are approximately equal to any given pattern.

Following are the advantages of using the Fuse.js library:

- Doesn't require a BE (dedicated backend) to handle the search functionality
- Performance is the prime target and criteria of this library.
- Mainly used when the datasets are moderately huge in size and for handling searching on the client-side where setting up a dedicated backend is not an option.

3.5 Frameworks Used:

Frameworks used in this project are as follows.

Axios: This is a promised (a concept in JS) based HTTP client for both node.js and the browser. The best part of this library is the fact that this is isomorphic, meaning it can be used not only to run in the browser but also in node.js while keeping the same codebase available. In our case, it is used to send and handle HTTP requests and responses using JS promises. Additionally, this is free and open source.

Vue: This is a progressive framework that is designed to build UI (user interfaces). It can be used as both a library and as a framework depending on the use case/goal. The core library focuses on the view layer. It also provides an ecosystem of supporting other libraries as well. This was mainly created as an alternative to the popular Angular.js, a frontend framework maintained by Google. We mainly chose Vue because it has the best features of angular as well as templating style like react. It is relatively easy and convenient to develop progressive web-apps which are built for scaling. Also when compared with other popular frontend libraries, it is also relatively better in terms of performance.

Express: The Express framework is a flexible Node.js framework that helps us with web applications and mobile applications. The foremost use of the Express framework is to help us with the server-side logic which helps us with the mobile and web applications. Few of the major advantages of using the express framework are that the developments we do on the web application can be done in a much faster and easier way. The framework can be easily configured and it is very easy to customize it.

3.6 Others:

JSON-LD: The JSON-LD is a format that is used to encode the linked data. This format allows the data to be numbered in such a fashion that it is very similar to the orthodox one. The format annotates content on a page and structures the data, which is then further used by search engines

Docker: The Docker platform is an open and free platform for creating and running software. This platform helps us to distinguish our applications from our infrastructure so we can deliver applications quickly. With this platform, we can manage our infrastructure in the same way we manage our applications.

3.7 Database Used:

SQLite3: This is an RDBMS (Relational Database management system) which is contained in the C library. Unlike most of the database systems available sqlite3 is not really a client-server database engine, rather this is embedded into the end program. The syntax style is similar to the one found in PostgresSQL. This was the database of our choice mainly because it is server-less and self contained unlike MySQL which requires a separate server to run, thereby demanding a client and server architecture to interact over any network..

Redis: This is an open-source database. It is very popular and widely used by developers for a variety of uses. It is mainly used for maintaining cache, as a message broker etc. Considering its extremely fast speed and easy to configure, we have used this for maintaining a task queue in the parsing section which we'll talk about in detail in the subsequent section.

3.8 Understanding the architecture:

3.8.1 Parsing Section

The parsing section accounts for parsing data and making it suitable for the use case. It includes scrapping of data, Format conversion, Data generation etc.

The parsing section is again sub-divided into 3 further sections:

- Web Crawler
- Web Scraper
- Parser

3.8.1.1 Web Crawler

We made a generalized script to crawl the web, it is a program written specifically to navigate through a list of web pages, herein Wikipedia pages, and do predefined tasks of finding related web pages (Wikipedia links).

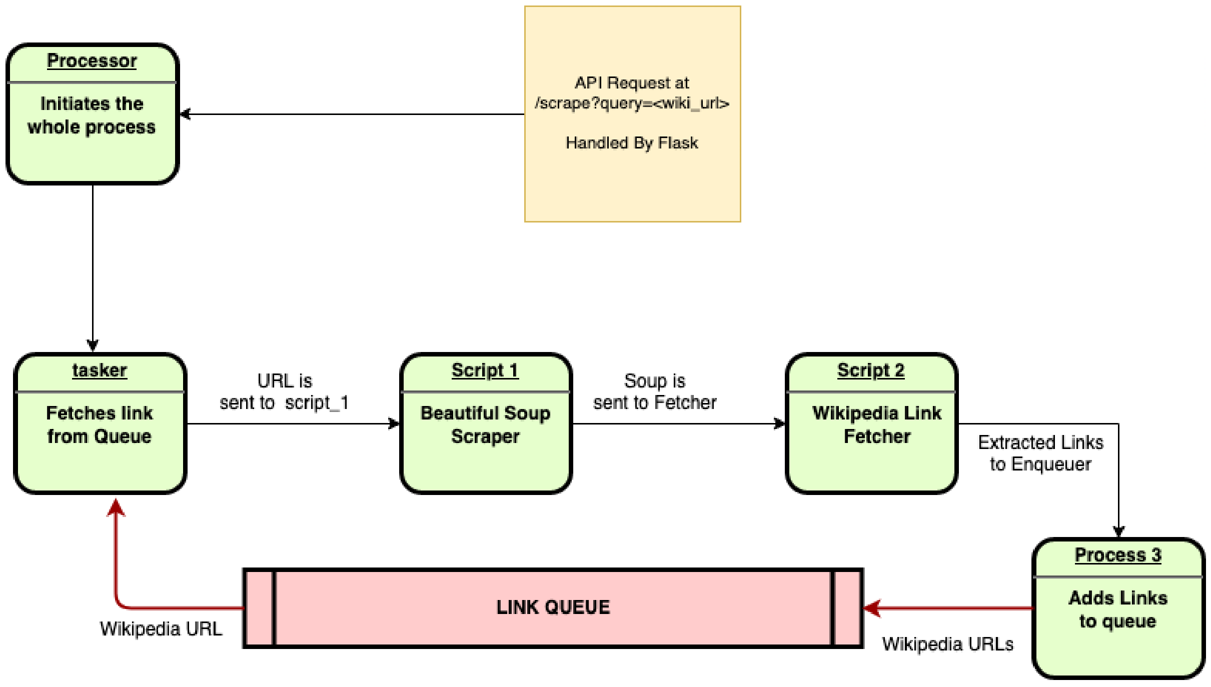
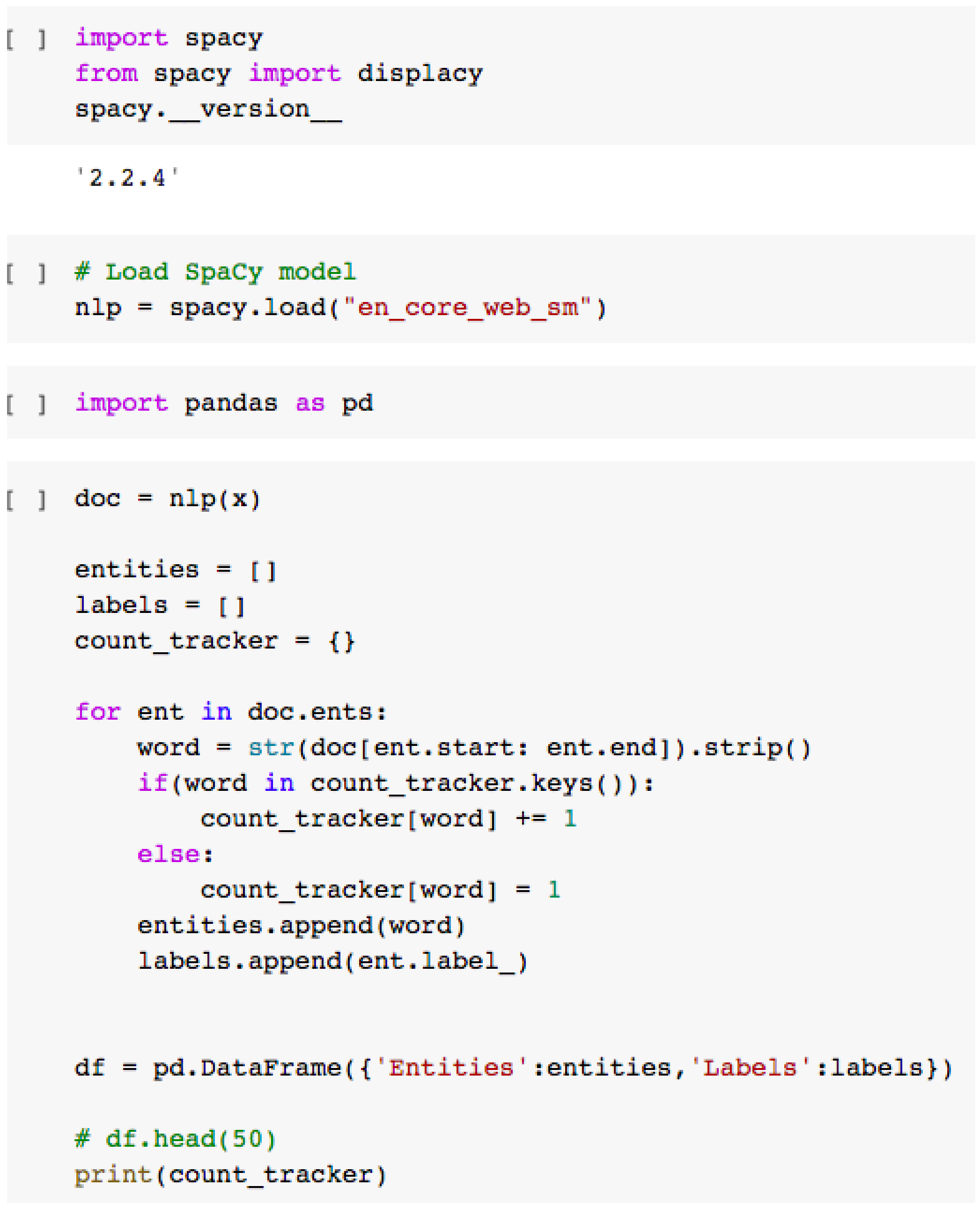


Fig 3.3 Flow of the crawler

3.8.1.2 Web Scraper

We made another script to scrape through these Wikipedia pages, we crawled earlier, to extract content and data using the beautiful soup library which is available in python. This scraped data was available to us in different formats ranging from texts, tables, and other files of various formats.



```
[ ]  import spacy
     from spacy import displacy
     spacy.__version__

     '2.2.4'
```

```
[ ]  # Load SpaCy model
     nlp = spacy.load("en_core_web_sm")
```

```
[ ]  import pandas as pd
```

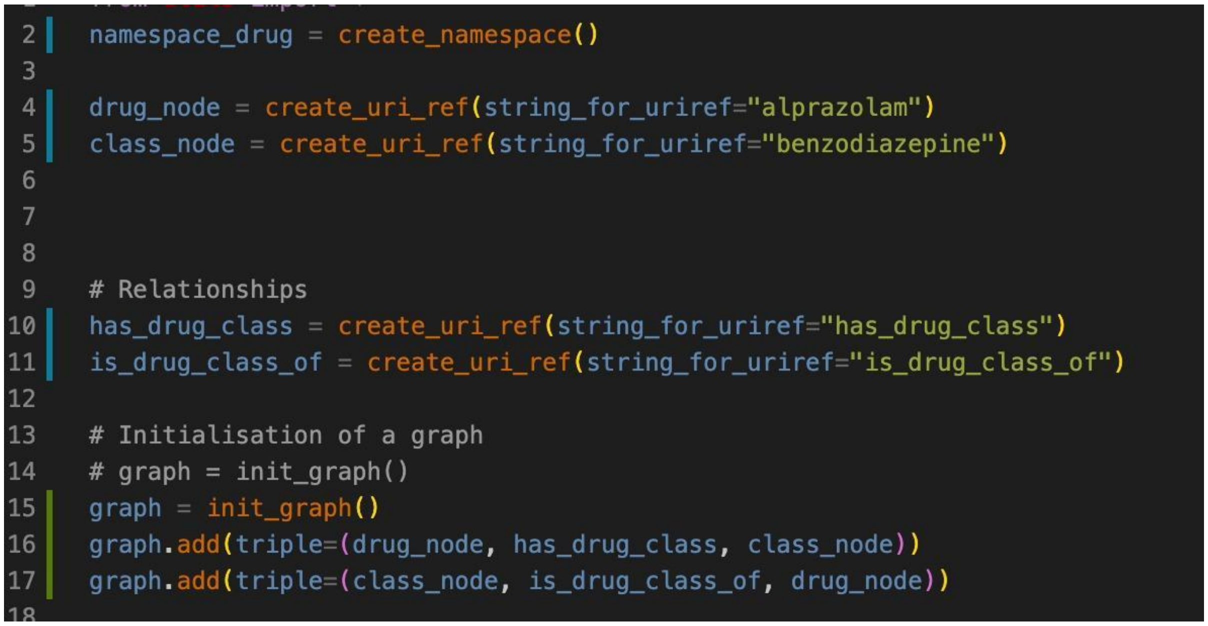
```
[ ]  doc = nlp(x)

     entities = []
     labels = []
     count_tracker = {}

     for ent in doc.ents:
         word = str(doc[ent.start: ent.end]).strip()
         if(word in count_tracker.keys()):
             count_tracker[word] += 1
         else:
             count_tracker[word] = 1
         entities.append(word)
         labels.append(ent.label_)


     df = pd.DataFrame({'Entities':entities,'Labels':labels})

     # df.head(50)
     print(count_tracker)
```

Fig 3.4 Code for extracting keywords using the spacy library.

3.8.1.3 Parser

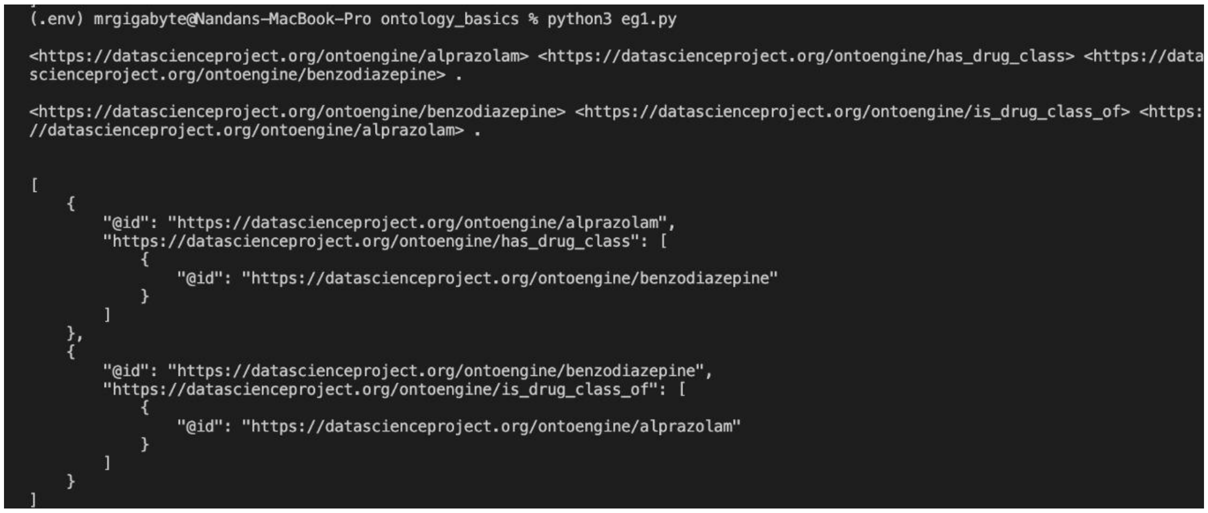
The data received through the scraper was further passed through a parsing engine, which sent a fraction of data which is tabular in nature directly to the database while the remaining data was used to create RDF triples. At this stage, the RDF triples can be converted into tabular data and feeded into the database if required and the tabular data in the database can also be converted into RDF triples as per requirement. Further, these RDF triples are converted into the JSON-LD format as the OntoEngine works only in JSON-LD format. The RDF library was used to create ontologies using the keywords we extracted doing the above process.

(a) Ontology Generator:



```
namespace_drug = create_namespace()

drug_node = create_uri_ref(string_for_uriref="alprazolam")
class_node = create_uri_ref(string_for_uriref="benzodiazepine")



# Relationships
has_drug_class = create_uri_ref(string_for_uriref="has_drug_class")
is_drug_class_of = create_uri_ref(string_for_uriref="is_drug_class_of")

# Initialisation of a graph
# graph = init_graph()
graph = init_graph()
graph.add(triple=(drug_node, has_drug_class, class_node))
graph.add(triple=(class_node, is_drug_class_of, drug_node))
```

Fig 3.5 Example of how ontologies are created using the custom functions and libraries.



```
(.env) mrgigabyte@Nandans-MacBook-Pro ontology_basics % python3 eg1.py

<https://datascienceproject.org/ontoengine/alprazolam> <https://datascienceproject.org/ontoengine/has_drug_class> <https://data
scienceproject.org/ontoengine/benzodiazepine> .

<https://datascienceproject.org/ontoengine/benzodiazepine> <https://datascienceproject.org/ontoengine/is_drug_class_of> <https:
//datascienceproject.org/ontoengine/alprazolam> .


[
    {
        "@id": "https://datascienceproject.org/ontoengine/alprazolam",
        "https://datascienceproject.org/ontoengine/has_drug_class": [
            {
                "@id": "https://datascienceproject.org/ontoengine/benzodiazepine"
            }
        ]
    },
    {
        "@id": "https://datascienceproject.org/ontoengine/benzodiazepine",
        "https://datascienceproject.org/ontoengine/is_drug_class_of": [
            {
                "@id": "https://datascienceproject.org/ontoengine/alprazolam"
            }
        ]
    }
]
```

Fig 3.6 Output of the created ontology

(b) Helper Functions and Algorithms :

create_uri_ref

Usage:

URIRef is used to define classes, properties or any entity needed for an Ontology.

Parameters Accepted:

string_for_uriref: Type String

namespace_prefix: Type String

Output Expected:

RDF URI

create_namespace

Usage:

This function creates a namespace that can be used in the rdf for linking other nodes to the function

Parameters Accepted:

suffix: Type String

namespace_prefix: Type String

Output Expected:

RDF Namespace

format_convert_graph

Usage:

This function converts ontologies from one form to the other based on the format and location specified.

Parameters Accepted:

source: Type String

Example - “drug_target.xrdf”

source_format: Type String

Example - “application/rdf+xml”

destination: Type String

Example - “json-ld”

destination_format: Type String
Example - “drug_target.json”

3.8.2 OntoEngine

Once the information we need is in the required format i.e. JSON-LD format, we can run queries through this engine. The features of the engine are as follows:

- Query An Ontology based on the predicate
- Query An ontology based on the type
- Query Multiple Ontologies together at the same time
- View Specific iNode information for the selected node.

To achieve this the OntoEngine part is again sub-divided into 2 further sections:

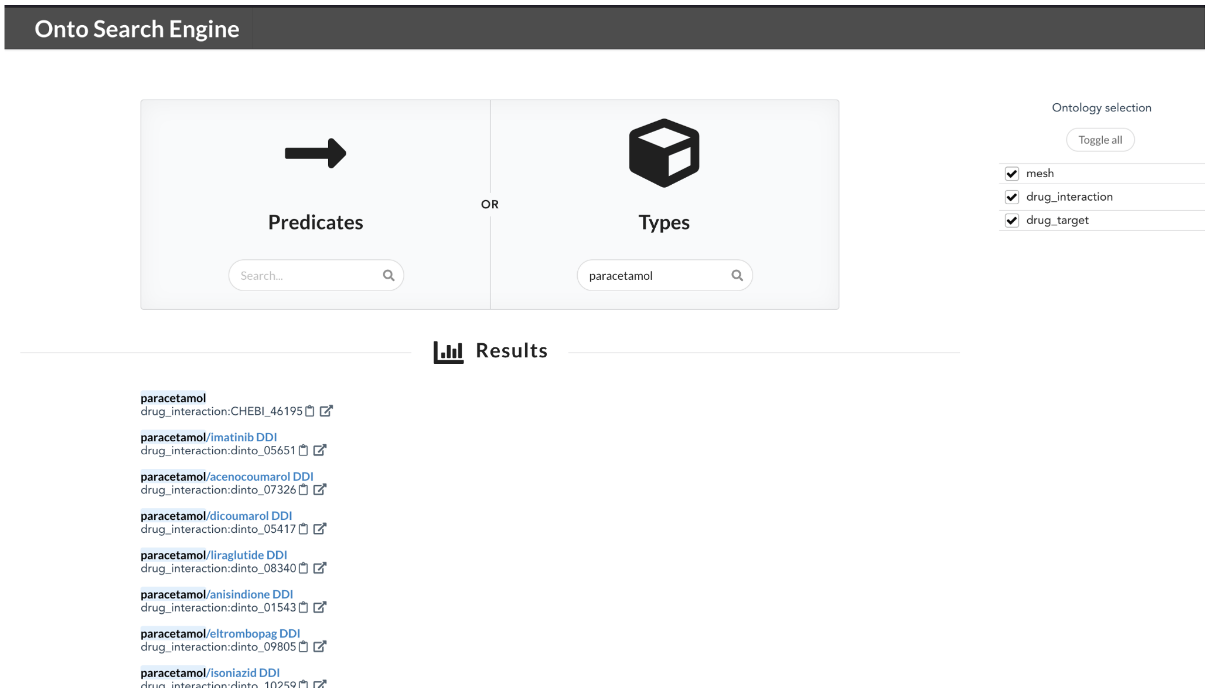
- Front end
- Back end

3.8.2.1 Front End

The front end of this project is made using Vue for making it an interactive and appealing interface. Vue as stated earlier in the previous section is basically a framework for building user interfaces. We chose Vue over React primarily because of the level of simplicity Vue offers when compared to React which is mainly suitable for building large scale applications.

The UI is simple and easy to use. The frontend contains 2 pages.

(a) Search Results Page : Querying section on the ontologies available, the sidebar on the right shows all the list of available ontologies while the main results section shows the available results. The user can search based on predicates OR types.
The Label and Comment predicates are shown in the result list. Detail section can be seen when specifically selecting a particular search result.

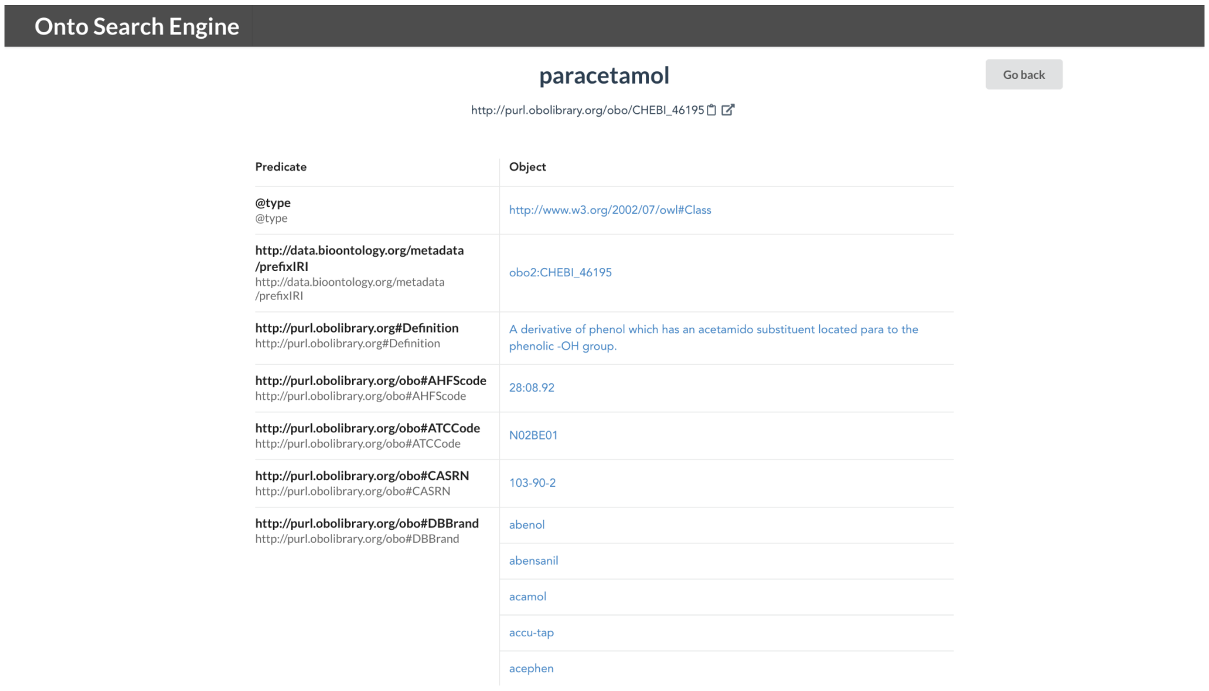


paracetamol
✓ mesh
✓ drug_interaction
✓ drug_target

paracetamol
drug_interaction:CHEBI_46195
paracetamol/imatinib DDI
drug_interaction:dinto_05651
paracetamol/acenocoumarol DDI
drug_interaction:dinto_07326
paracetamol/dicoumarol DDI
drug_interaction:dinto_05417
paracetamol/liraglutide DDI
drug_interaction:dinto_08340
paracetamol/anisindione DDI
drug_interaction:dinto_01543
paracetamol/eltrombopag DDI
drug_interaction:dinto_09805
paracetamol/isoniazid DDI
drug_interaction:dinto_10259

Fig 3.7 Screenshot of Search Results Page

(b) Node Information Page : This page shows the information of a particular selected result, it's mainly a table containing properties and their values. The user can even be redirected to the actual entity by clicking on the hyperlink available. This is useful when you wish to query your dataset and explore.


Onto Search Engine

Go back

paracetamol

http://purl.obolibrary.org/obo/CHEBI_46195

Predicate	Object
@type @type	http://www.w3.org/2002/07/owl#Class
http://data.bioontology.org/metadata/prefixIRI http://data.bioontology.org/metadata/prefixIRI	obo2:CHEBI_46195
http://purl.obolibrary.org#Definition http://purl.obolibrary.org#Definition	A derivative of phenol which has an acetamido substituent located para to the phenolic -OH group.
http://purl.obolibrary.org/obo#AHFScode http://purl.obolibrary.org/obo#AHFScode	28:08.92
http://purl.obolibrary.org/obo#ATCCode http://purl.obolibrary.org/obo#ATCCode	N02BE01
http://purl.obolibrary.org/obo#CASRN http://purl.obolibrary.org/obo#CASRN	103-90-2
http://purl.obolibrary.org/obo#DBBrand http://purl.obolibrary.org/obo#DBBrand	abenol
	abensanil
	acamol
	accu-tap
	acephen

Fig 3.8 Screenshot of Node Information Page

3.8.2.1 Back End

The backend of this project is created using NodeJS (Express).
It consists of routes. Every request to the backend is eventually executed by a controller. A set of routes are defined which further maps the given request to the controller. Routes are defined inside the extensions of the file.

Routes

GET /api/:searchType -

<u>Parameters:</u>

searchType : Type String, accepts “predicate” & “type”

term : Type String, accepts a search value

<u>Usage:</u>

Used for querying ontologies based on given value

<u>Results:</u>

Success: sends the available search results

Failure:

400 : Search Type not defined,

400 : Search term not defined

400 : No ontology was selected

500 : error caught

GET /api/ontology-list -

Parameters:

NA

Usage:

This route is used to fetch the available ontologies in the database

Results:

Success: JSON containing key value pairs of ontology name and their URI

Failure:

500: error caught

GET /api/result -

Parameters:

searchType : Type String, accepts “predicate” & “type”

uri : Type String, accepts the unique reference identifier

Usage:

This is used for retrieving information for a particular URI.

Results:

Success: sends the available search results

Failure:

500 : Search Type not defined

500 : URI is not defined

500 : error caught

POST /api/feedback -

Parameters:

feedback : Type String, accepts the user feedback

Usage:

Used for accepting the user feedback at any given point in time.

It stores the feedback in the SQLite3 database.

Results:

Success:

200 : Success

Failure:

500 : error caught

Helper Functions and Algorithms :

prepareData

Usage:

This function basically as the name suggests prepares the Ontologies available in the database to be served on the endpoint when the results are requested.

Parameters Accepted:

ontology:

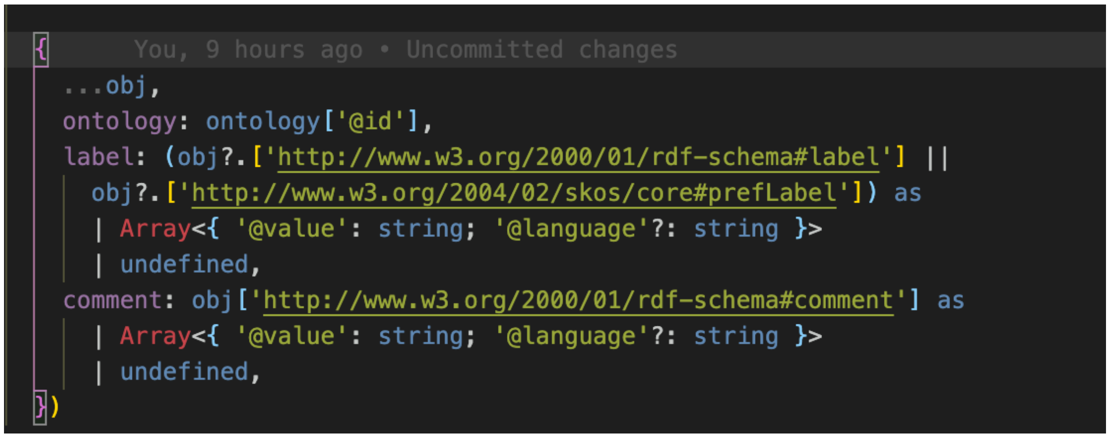
Type { '@id' : String, '@graph' : Array<Obj> }

Example:

```
{
        "@id" : "https://example/com",
        "@graph" : [
                "https://w3.com/owl#Label" : {
                                "@value" : "test"
                }
        ]
}
```

Output Expected:

Ontology with each object containing Label and Comment additional keys



```
{       You, 9 hours ago • Uncommitted changes
  ...obj,
  ontology: ontology['@id'],
  label: (obj?.['http://www.w3.org/2000/01/rdf-schema#label'] ||
    obj?.['http://www.w3.org/2004/02/skos/core#prefLabel']) as
    | Array<{ '@value': string; '@language'?: string }>
    | undefined,
  comment: obj['http://www.w3.org/2000/01/rdf-schema#comment'] as
    | Array<{ '@value': string; '@language'?: string }>
    | undefined,
})
```

Fig 3.9 return object of prepareData function

mergedOntologies

Usage:

This is a helper function it is used to merge all the available ontologies and prepares them to be queries upon. It basically returns a promise containing an array of available ontologies in the database.

Parameters Accepted:

NA

Output Expected:

This returns a promise containing an array of all the ontologies.

Ontologies List : Promise(Array<Object>)

prepareIndex

Usage:

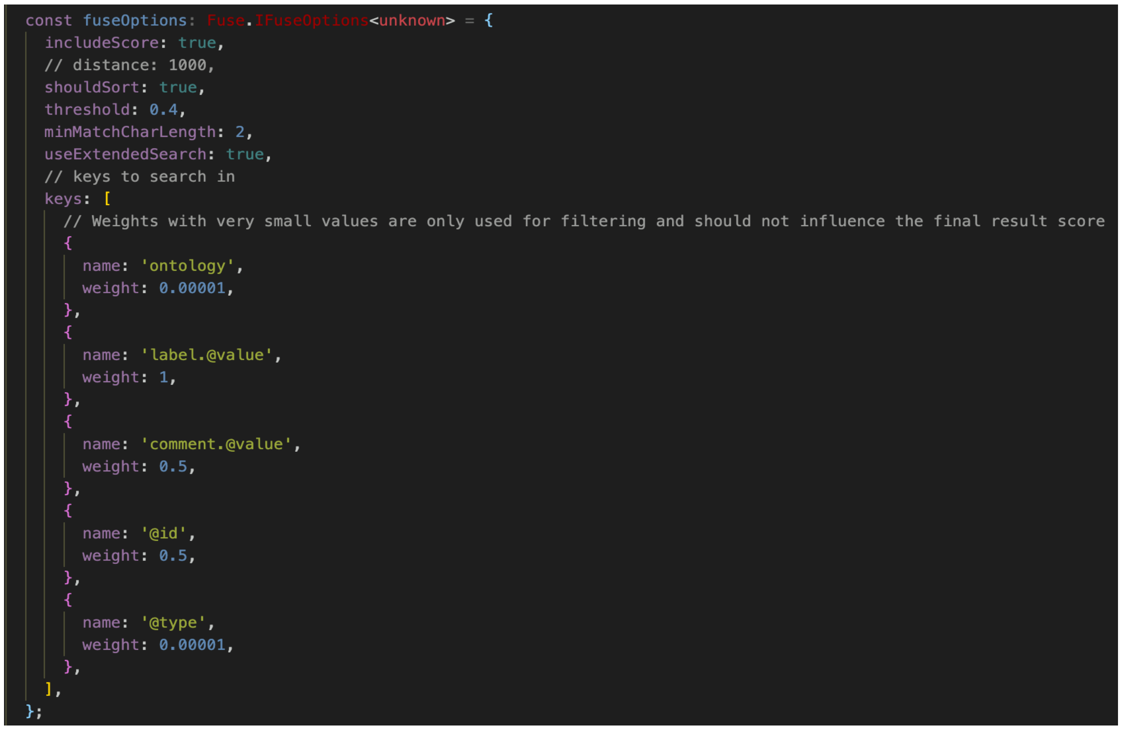
This is an important helper function it is basically calls the mergedOntology function and basically uses the Fuse.js for indexing. The fuse options are configured here which helps with the elastic search. The weights are also set here for the search results. The fuse instance is basically updated with the loaded indexed data.

Parameters Accepted:

NA

Output Expected:

NA



```
const fuseOptions: Fuse.IFuseOptions<unknown> = {
  includeScore: true,
  // distance: 1000,
  shouldSort: true,
  threshold: 0.4,
  minMatchCharLength: 2,
  useExtendedSearch: true,
  // keys to search in
  keys: [
    // Weights with very small values are only used for filtering and should not influence the final result score
    {
      name: 'ontology',
      weight: 0.00001,
    },
    {
      name: 'label.@value',
      weight: 1,
    },
    {
      name: 'comment.@value',
      weight: 0.5,
    },
    {
      name: '@id',
      weight: 0.5,
    },
    {
      name: '@type',
      weight: 0.00001,
    },
  ],
};
```

Fig 3.10 Fuse Configuration for Elastic Search Results.

getFromURI

Usage:

This is basically used to fetch information based on the particular input URI. The algorithm is straightforward. It basically queries on the earlier created Fuse Instance with the input URI

Parameters Accepted:

searchType : Type String, accepts “predicate” & “type”

searchUri : Type String, accepts the URI

Output Expected:

The required search object containing all the information of the input URI

search

Usage:

This is the main search algorithm which works on the created Fuse instance when we call the prepareIndex function. Based on the given input it returns the available search results.

Parameters Accepted:

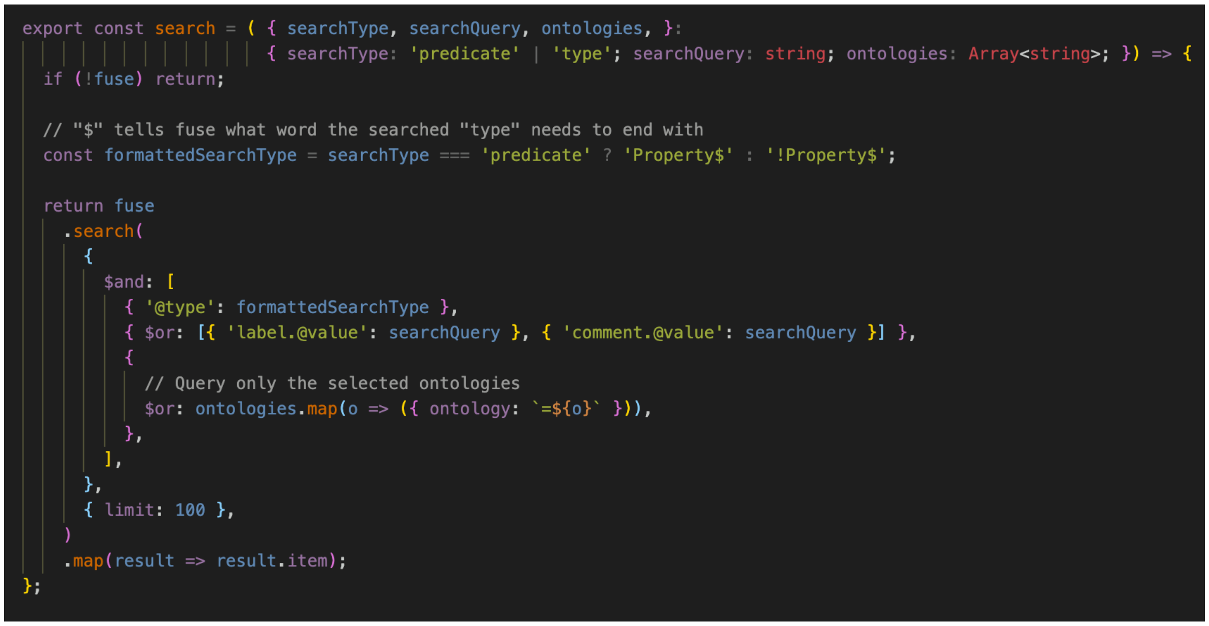
searchType : Type String, accepts “predicate” & “type”

searchQuery : Type String, accepts the search query value

Ontologies : Type Array<String>. array of ontologies to query on

Output Expected:

______Array of Objects containing matching entities.



```
export const search = ( { searchType, searchQuery, ontologies, }:
                        { searchType: 'predicate' | 'type'; searchQuery: string; ontologies: Array<string>; }) => {
  if (!fuse) return;

  // "$" tells fuse what word the searched "type" needs to end with
  const formattedSearchType = searchType === 'predicate' ? 'Property$' : '!Property$';

  return fuse
    .search(
      {
        $and: [
          { '@type': formattedSearchType },
          { $or: [{ 'label.@value': searchQuery }, { 'comment.@value': searchQuery }] },
          {
            // Query only the selected ontologies
            $or: ontologies.map(o => ({ ontology: `=${o}` })),
          },
        ],
      },
      { limit: 100 },
    )
    .map(result => result.item);
};
```

Fig 3.11 Search Function

importNqFromPrefix

Usage:

This is a helper function that returns a file instance based on the ontology names. This is basically used to import an ontology based on the name provided.

Parameters Accepted:

prefix : Type String, accepts ontology name

Output Expected:

______File instance of the data based on the input prefix

nQuadsToJsonLd

Usage:

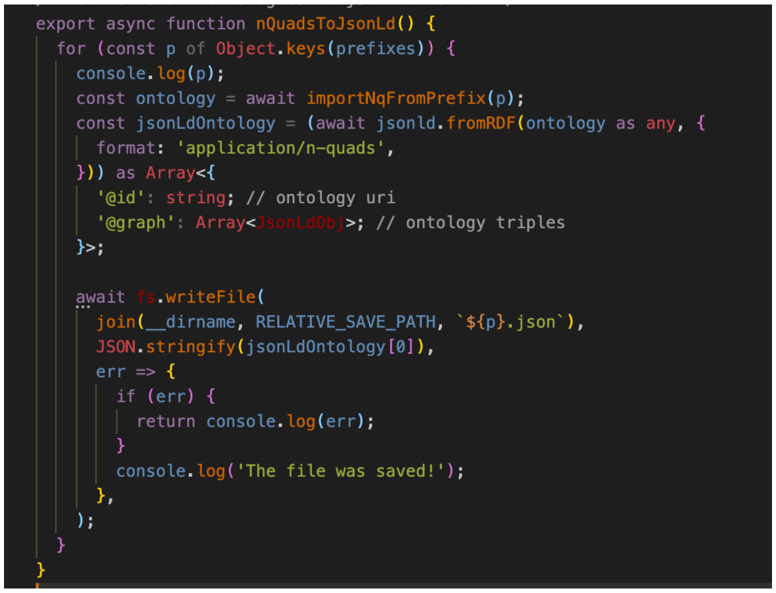
This is also an important algorithm for converting nQuands Type to JSON Ld format which is what we are using for storing it in the database. The algorithm is pretty simple and straightforward, it takes in the list of available ontologies, calls the fromRDF function of jsonLd library and converts it into the required JSON Ld format. After successful conversion a new JSON file is created containing the newly converted data into JSON Ld.

Parameters Accepted:

NA

Output Expected:

______NA



```
export async function nQuadsToJsonLd() {
  for (const p of Object.keys(prefixes)) {
    console.log(p);
    const ontology = await importNqFromPrefix(p);
    const jsonLdOntology = (await jsonld.fromRDF(ontology as any, {
      format: 'application/n-quads',
    })) as Array<{
      '@id': string; // ontology uri
      '@graph': Array<JsonLdObj>; // ontology triples
    }>;

    await fs.writeFile(
      join(__dirname, RELATIVE_SAVE_PATH, `${p}.json`),
      JSON.stringify(jsonLdOntology[0]),
      err => {
        if (err) {
          return console.log(err);
        }
        console.log('The file was saved!');
      },
    );
  }
}
```

Fig 3.12 nQuadsToJsonLd function

3.8.3 Database

The whole model is supported by different databases. The choice of the database mainly depends on the functionality and requirements. Here's a list of databases used:

Redis:

It's a robust in-memory NoSQL database platform which is extremely fast and convenient to use. This has been used by companies throughout to maintain their cache. In our use case we are using redis to actively maintain a redis queue. While parsing, each newly scrapped link is added to the queue which will be later fed to the scrapper for data extraction.

SQLite3:

This is used in the OntoEngine Portion for solely maintaining any input feedback by the user while using the interface. It is not used for anything else.

JSON Files:

While this isn't an actual database, the JSON files are used as source of data to be queried on. All the ontologies that we have used are converted and stored in the form of JSON which can then be actively queried upon as per the need.

CHAPTER - 4

PERFORMANCE ANALYSIS

As discussed in the previous chapter, the entire model is broken down into various sections, we have analysed and checked for performance in each of the sections. Moreover a thorough comparison has also been done in each section with the already existing model / technology / framework in the market.

4.1 Parsing Section:

This section consists of the following libraries and algorithms:

1. BeautifulSoup :

    Functionality :

        For scraping and parsing web-pages.

    Performance :

        It was observed that the download of a particular web-page takes about 1-2 seconds, while accounting for high network latency.
        We moreover used a profiler to debug for the performance.
        We further compared the performance using two parsers:

            (a) http.parser: This is a build in HTML parser in python 3
            (b) lxml : a parser written in C

        It was observed that the http.parser took 6.85 seconds to parse / scrape through 4 pages. Whereas it did the same thing in 0.6 seconds.

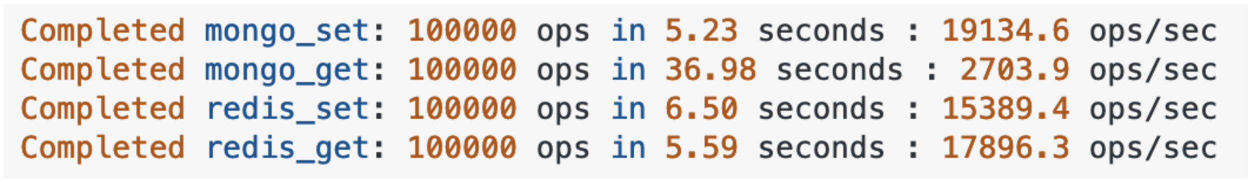
2. Redis Queue :

Functionality :

For storing all active links to be crawled and scrapped by maintaining a queue.

Performance :

Our goal for picking redis is primarily because of the fast speed it offers right out of the box. We performanced a series of benchmark tests on redis and compared it with the already existing and popular database system, mongoDB which is used for storing the similar information like ours and the result was as follows:



```
Completed mongo_set: 100000 ops in 5.23 seconds : 19134.6 ops/sec
Completed mongo_get: 100000 ops in 36.98 seconds : 2703.9 ops/sec
Completed redis_set: 100000 ops in 6.50 seconds : 15389.4 ops/sec
Completed redis_get: 100000 ops in 5.59 seconds : 17896.3 ops/sec
```

Fig 4.1 It can be observed that redis is about 7x faster when performing the get operation as compared to mongo DB.

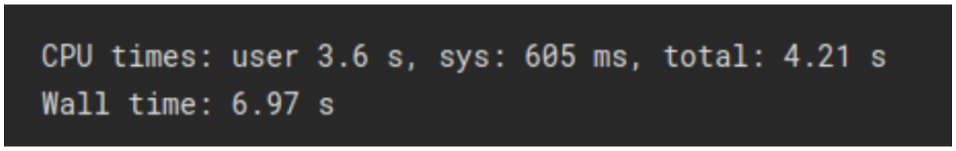
3. Dask :

Functionality :

For parsing through already existing datasets in different formats and performing operations on them.

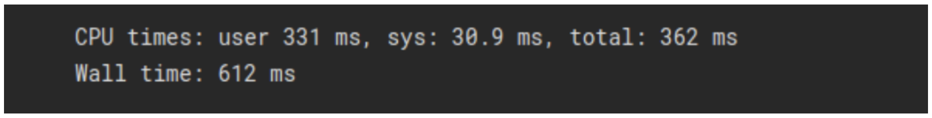
Performance :

Dask, as stated in the earlier sections, is used for parallel computation in python. Pandas is another popular alternative for Dask but it comes with a limitation: it isn't suitable for handling extremely large datasets. Hence, this is where dask comes in and uses the power of dynamic task scheduler and performs the required task in way less time. We did an analysis to compare dask with widely used pandas and the results were as follow:



```
CPU times: user 3.6 s, sys: 605 ms, total: 4.21 s
Wall time: 6.97 s
```

(a)



```
CPU times: user 331 ms, sys: 30.9 ms, total: 362 ms
Wall time: 612 ms
```

(b)

Fig 4.2 Time taken by (a) Pandas and (b) Dask to read a particular pharma dataset

4. Spacy :

<u>Functionality</u> :

For annotating the words scrapped from the wiki articles and categorising them.

<u>Performance</u> :

NLTK and Spacy are mainly the two popular libraries when it comes to performing various operations on textual content. We did an analysis and compared the two libraries with each other.

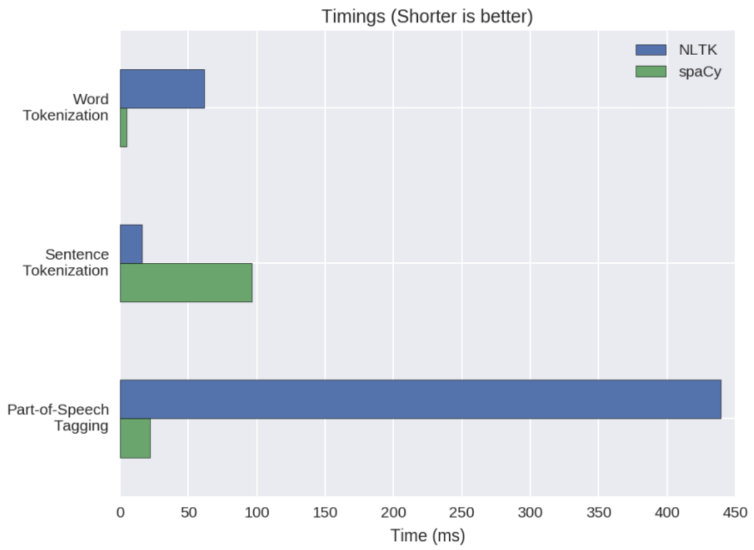


Fig 4.3 Comparison of NLTK vs Spacy

It was observed that Spacy performed better than NLTK when it comes to word tokenization, but the case wasn't the same in sentence tokenization this is mainly due to the fact that spacy uses syntactic tree approach for tokenizing sentences whereas NLTK splits the text into sentences.

4.2 OntoEngine Section:

This section consists of the following libraries and algorithms:

1. SQLite3 :

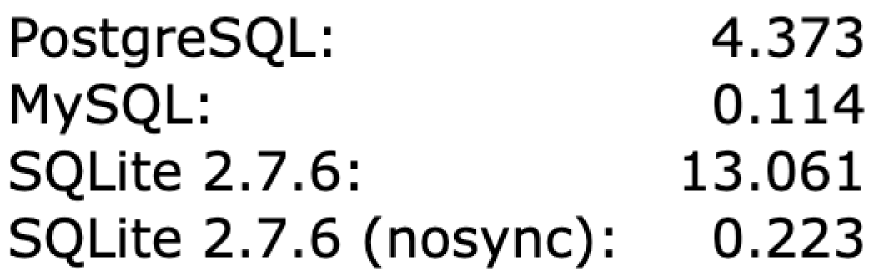
Functionality :

Used for storing metadata and extra user information.

Performance :

The choice of this database management system was mainly because it is very easy to set up and containerized. We performed a series of operations on SQLite3 vs some other handpicked database management systems and compared the results with each other.

We performed 1000 Insertions and this was observed :



PostgreSQL:	4.373
MySQL:	0.114
SQLite 2.7.6:	13.061
SQLite 2.7.6 (nosync):	0.223

Fig 4.4 Performance Analysis of SQLite with other database management systems

Since we are not using SQLite3 extensively as we have mainly used it for retrieval of metadata and user information, we felt that the difference in performance doesn't have a huge impact on the interface as of now.

2. Search (function: Backend):

Functionality :

Function used for querying on the ontology and returning matching results

Performance :

This is basically a fuzzy-search implementation that iterates through the JSON files that we have as ontology data and returns results which are appropriate. When checking for performance we performed a search on about 250,000 data-nodes / entities, and it performed the operation without any hitch, we used production build in this case. Sometimes however, we did notice some small issues in terms of performance on the development build, but so far it ran smoothly in most cases.

CHAPTER - 5

CONCLUSIONS

5.1 Conclusions

After doing an in depth research in regards to the semantic analysis in the pharmaceutical domain, the fuzzy search engine helped us in querying through information we require across various databases despite the data sets being irregular in nature. The idea of having various information we require on one page helped us in increasing efficiency of the work and also helped us in saving time. Not only this we were successful in converting the text data to ontologies which represented the information in the form of nodes and relationships. This proves useful when one aims on finding all possible information surrounding a particular entity of concern, in this case a drug (example: Paracetamol)

5.2 Future Scope

Semantic data is something that was introduced as a concept a long time ago (1990s). It however lacks any progress of evolution for quite some time. In fact, 2007 was the last boom in the technology with the introduction of OWL in the manner of representing semantic information in an ontology while also providing more forms of expression through Description Logic (DL). However, the applications required to develop an Ontology and any form of DL expression such as editors for Ontologies etc are all currently in their infancy and/or outdated/complex design editors.

Semantic Analysis on the other hand is a very new concept specifically to build search and query mechanisms using inferences through reasonsers perhaps with the right support and aid

we think that we can be successful in terms of creating a framework that can help support and extend the power of semantic data and perform various operations on it.
In Future, we believe that it can be used as a search engine which can ingest large volumes of information specific to a project/research study and represent them as graphs as well as allowing for the implementation of faster search algorithms, making mind maps easier, one could also extend this to have various forms of structure for learning.

REFERENCES

1. Forum on Neuroscience and Nervous System Disorders, "Drug Development Challenges," *Improving and Accelerating Therapeutic Development for Nervous System Disorders: Workshop Summary.*, 06-Feb-2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK195047/. [Accessed: 05-Dec-2021]
2. "36 Web Resources For Target Hunting In Drug Discovery," *BioPharmaTrend*. [Online]. Available: https://www.biopharmatrend.com/post/45-27-web-resources-for-target-hunting-in-drug-discovery/. [Accessed: 05-Dec-2021].
3. "Pharmacy," *Wikipedia*, 28-Nov-2021. [Online]. Available: https://en.wikipedia.org/wiki/Pharmacy. [Accessed: 05-Dec-2021].
4. C. Katiyar, A. Gupta, S. Kanjilal, and S. Katiyar, "Drug discovery from plant sources: An integrated approach," *Ayu*, Jan-2012. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3456845/. [Accessed: 05-Dec-2021].
5. 12 May 2015 By Ingrid Torjesen, "Drug development: the journey of a medicine from lab to shelf," *The Pharmaceutical Journal*, 12-Feb-2021. [Online]. Available: https://pharmaceutical-journal.com/article/feature/drug-development-the-journey-of-a-medicine-from-lab-to-shelf. [Accessed: 05-Dec-2021].
6. J. H. Ramirez and M. Langan, "Drug testing database," *figshare*, 19-Jan-2016. [Online]. Available: https://figshare.com/articles/dataset/Drug_testing_database/1270272. [Accessed: 05-Dec-2021].
7. "Homepage," *ECHA*. [Online]. Available: https://echa.europa.eu/. [Accessed: 05-Dec-2021].
8. Forum on Neuroscience and Nervous System Disorders, "Drug Development Challenges," *Improving and Accelerating Therapeutic Development for Nervous*

System Disorders: Workshop Summary., 06-Feb-2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK195047/. [Accessed: 05-Dec-2021].

9. "Food and Drug Administration," *Wikipedia*, 07-Nov-2021. [Online]. Available: https://en.wikipedia.org/wiki/Food_and_Drug_Administration. [Accessed: 05-Dec-2021].