# Information Retrieval Resit Project: Medications Search Engine

Hoormazd Pirayeshfar

September 2023

## 1 Introduction

This report outlines the development of a medication search engine, which encompasses web scraping, data indexing, and a user interface creation process. Our objective was to enable users to search for specific medications and retrieve comprehensive information from reputable sources, including drugs.com, rxlist.com, and webmd.com.

### 1.1 Project Overview

1. Web Scraping: We initiated the project by building a web scraper using Scrapy. This tool extracted data from three chosen websites: drugs.com, rxlist.com, and webmd.com, systematically gathering details about drugs, such as names, uses, side effects, and more.

2. Data Indexing: To optimize search results, we employed Apache Solr to index the scraped medication data. This process organized the information for efficient retrieval, leveraging Apache Solr's robust text search capabilities.

3. User Interface: The final stage involved creating a user-friendly web interface for medication searches. Users can easily find specific medications by name, category, or other criteria, ensuring a seamless and intuitive experience.

Throughout this report, we will delve into technical details, including tools, challenges, and solutions.

## 2 Crawling and Scraping

As mentioned in the introduction we have three chosen websites: 1. drugs.com 2. rxlist.com 3. webmd.com

The useful 'fields' that all of these resources had in common were the following: name, generic information, description, warnings, side effects, and usage instructions. Therefore, I decided to create a 'MedicationItem' within the scrapy items.py structure with the following fields to have a unified system to work with all these different websites:

```
import scrapy
class MedicationItem(scrapy.Item):
    id = scrapy.Field()
    source = scrapy.Field()
    source_url = scrapy.Field()
    name = scrapy.Field()
    generic_name = scrapy.Field()
    description = scrapy.Field()
    warnings = scrapy.Field()
    side_effects = scrapy.Field()
    how_to_take = scrapy.Field()
    pass
```

Now all there is to do is to create 'MedicationItem's by crawling each website and then feed them down the pipeline.

The pipelines.py is configured to take these medication items and save them in both json and csv formats (mostly because at the time of doing this I didn't really know which format does Solr accept).

Since each of our websites has a different structure, I created three different spiders, each one designed to scrape information from a specific website, but since the items and pipelines were previously defined, all spiders will output data in the same unified format.

A huge problem however, was scraping drug information off of drugs.com as they had multiple different page designs and CSS formats which made it very difficult to scrape useful information. In the drug.com spider, (medication.py in the spiders folder) I accounted for two different types of CSS syntax used by drugs.com however it is still very unreliable because some of the pages were handwritten and they don't follow any specific rule.

After I figured I will be using the CSV output with Solr (see next section) I decided to separate the outputs of the 3 different spiders into 3 CSV files. The spiders scraped around 500 drugs from drugs.com, around 4000 from rxlist.com, and around 6000 from webmd.com; which brings the total document count to around 10500.

## 3 Solr: Indexing and Query Handling

I started with creating a new Solr core named 'medications' by duplicating the default configset, which made setting it up pretty easy. Since the scrapy output I decided to use were stored as a three CSV files (one for each website), I just uploaded them through Solr's document uploader, which I got to by running Solr:

```
./bin/solr start
```

And then navigating to:

```
http://localhost:8983/solr/#/medications/documents
```

From there, I could send send queries through http requests and receive matching documents from Solr.

The best way to index and find documents was to look for either matching names, generic info, or descriptions. Therefore, those are the only fields that queries will be send to.

No ranking was implemented for this because quite frankly I couldn't figure out what makes one medicine more important than another when given a query, however, I guess everything turned out pretty well at the end using the default config anyway.

## 4 User Interface and Retrieval

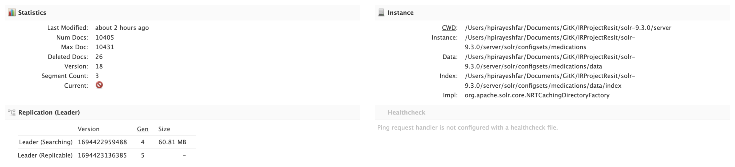
For this part I created a very simple HTML page with client-sided Javascript that has a single search bar. As the content of the search bar changes, the Javascript code will send a fetch GET request to Solr, with the following query to find matching results:

```
const response =
await fetch(`http://localhost:8983/solr/medications/select?
    q=name:*${query}* OR
    description:*${query}* OR
    generic_name:*${query}*
&rows=500`);
```

Even without a ranking system this works pretty well because typically users will either start entering the name of a drug, which will then be found in either the name field or the generic info field (named generic name in the code), or they enter a partial description which in this case will also return results. (e.g. pain, headache, ...)

## 5 Overview of the project

Solr statistics:



**Statistics**

| | |
|---|---|
| Last Modified: | about 2 hours ago |
| Num Docs: | 10405 |
| Max Doc: | 10431 |
| Deleted Docs: | 26 |
| Version: | 18 |
| Segment Count: | 3 |
| Current: | 🚫 |

**Replication (Leader)**

| | Version | Gen | Size |
|---|---|---|---|
| Leader (Searching) | 1694422959488 | 4 | 60.81 MB |
| Leader (Replicable) | 1694423136385 | 5 | - |

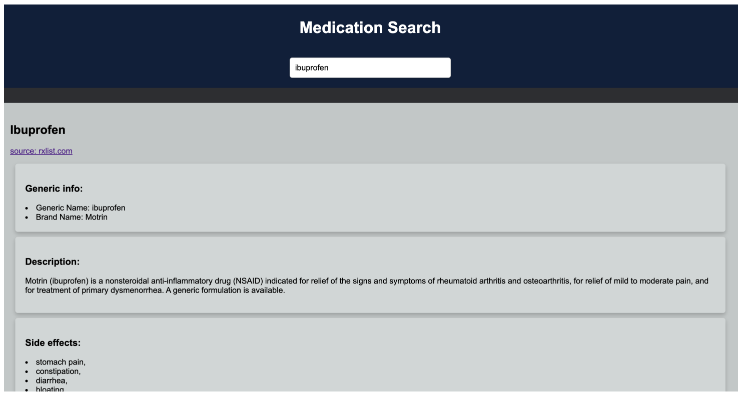
**Instance**

| | |
|---|---|
| CWD: | /Users/hpirayeshfar/Documents/GitK/IRProjectResit/solr-9.3.0/server |
| Instance: | /Users/hpirayeshfar/Documents/GitK/IRProjectResit/solr-9.3.0/server/solr/configsets/medications |
| Data: | /Users/hpirayeshfar/Documents/GitK/IRProjectResit/solr-9.3.0/server/solr/configsets/medications/data |
| Index: | /Users/hpirayeshfar/Documents/GitK/IRProjectResit/solr-9.3.0/server/solr/configsets/medications/data/index |
| Impl: | org.apache.solr.core.NRTCachingDirectoryFactory |

**Healthcheck**

Ping request handler is not configured with a healthcheck file.

Search results example:



**Medication Search**

ibuprofen

**Ibuprofen**

source: rxlist.com

**Generic info:**

- Generic Name: ibuprofen
- Brand Name: Motrin

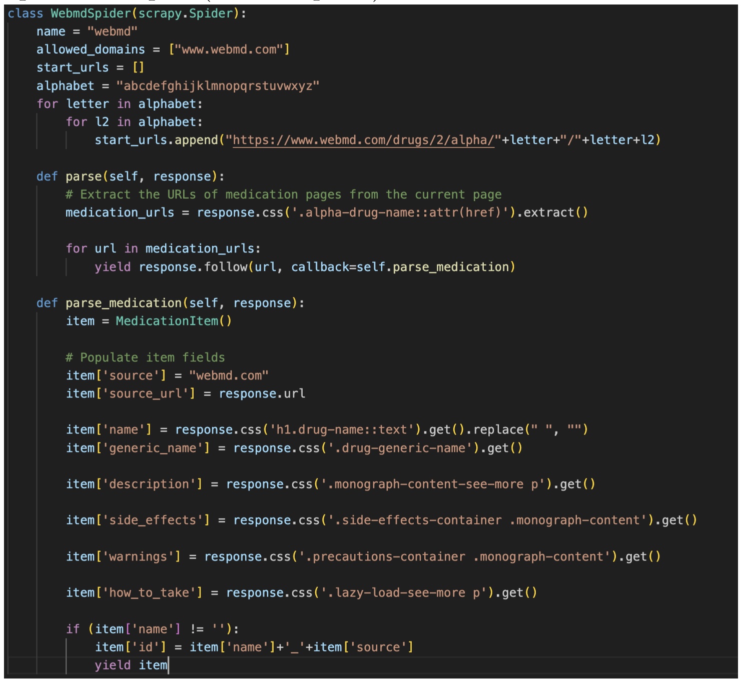
**Description:**

Motrin (ibuprofen) is a nonsteroidal anti-inflammatory drug (NSAID) indicated for relief of the signs and symptoms of rheumatoid arthritis and osteoarthritis, for relief of mild to moderate pain, and for treatment of primary dysmenorrhea. A generic formulation is available.

**Side effects:**

- stomach pain,
- constipation,
- diarrhea,
- bloating,

Spider example: (webmd spider)



```
class WebmdSpider(scrapy.Spider):
    name = "webmd"
    allowed_domains = ["www.webmd.com"]
    start_urls = []
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    for letter in alphabet:
        for l2 in alphabet:
            start_urls.append("https://www.webmd.com/drugs/2/alpha/"+letter+"/"+letter+l2)

    def parse(self, response):
        # Extract the URLs of medication pages from the current page
        medication_urls = response.css('.alpha-drug-name::attr(href)').extract()

        for url in medication_urls:
            yield response.follow(url, callback=self.parse_medication)

    def parse_medication(self, response):
        item = MedicationItem()

        # Populate item fields
        item['source'] = "webmd.com"
        item['source_url'] = response.url

        item['name'] = response.css('h1.drug-name::text').get().replace(" ", "")
        item['generic_name'] = response.css('.drug-generic-name').get()

        item['description'] = response.css('.monograph-content-see-more p').get()

        item['side_effects'] = response.css('.side-effects-container .monograph-content').get()

        item['warnings'] = response.css('.precautions-container .monograph-content').get()

        item['how_to_take'] = response.css('.lazy-load-see-more p').get()

        if (item['name'] != ''):
            item['id'] = item['name']+'_'+item['source']
            yield item
```

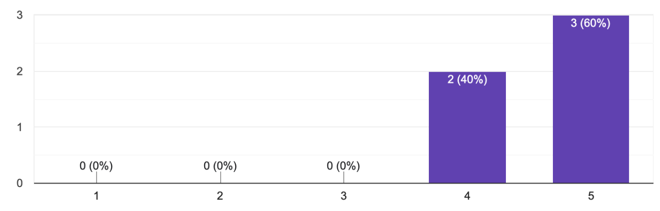
## 6 Testing

For user testing and feedback, I created a google form and asked a few friends to share their thoughts on the platform. You can find the results below:

Please search for a drug of your choice - are you satisfied with the results? How relevant were the results to what you were looking for? (5 - very good, 1 - very bad)

Copy

5 responses



What drug did you search for?

5 responses

Acetominophen

ketamine

ibuprofen

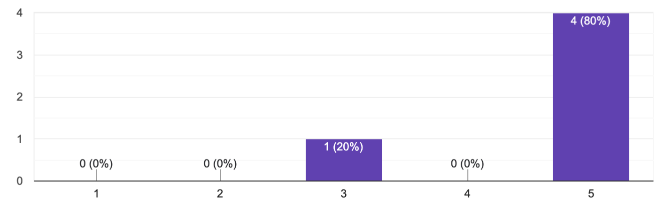
Pfizer-BioNTech COVID-19

Amoxicillin

Do you like the live search feature? Results are presented to you while typing.

Copy

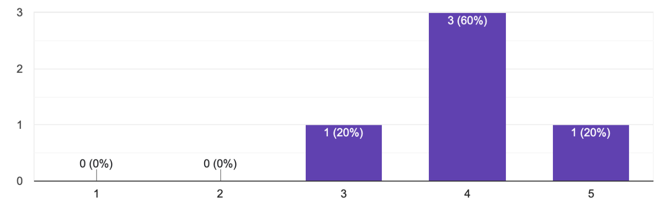
5 responses

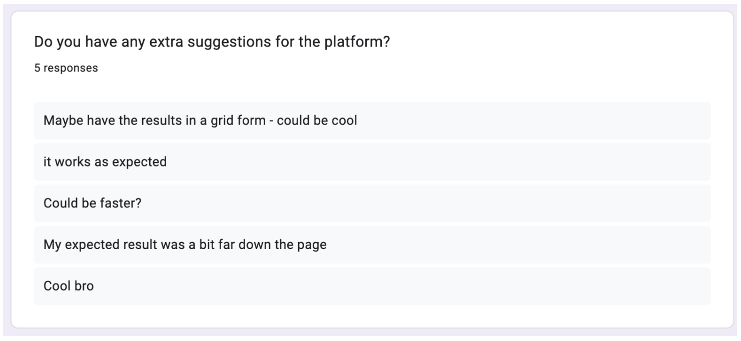


Give a general score for the whole patform. (5 - very good, 1 - very bad)

Copy

5 responses





Do you have any extra suggestions for the platform?

5 responses

Maybe have the results in a grid form - could be cool

it works as expected

Could be faster?

My expected result was a bit far down the page

Cool bro