CS699 Project

Parismita Das   Vaibhav Singh
Roll No: 22M0815   Roll No: 22M0827

23 November 2022

Drugpedia

Contents

Abstract

According to one's definition, the purpose of medicine is "to cure sometimes, to relieve often, to comfort always."[1]. Every medicine has its ingredient list thoroughly checked and verified, but despite that, some detrimental ingredients may be present that the patient is unaware of, or the patient is allergic to. One can look up these ingredients online, doing so will take a lot of efforts as there aren't many platforms that give this data collectively and one might not be able to find any certified information while surfing through the net. Our aim is to implement a platform where a user can find all the details related to the prescribed medicine and its ingredient compositions and their potential roles.

1 Problem Description

Drugpedia aims to incorporate Web Scraping to fetch relevant details regarding a medicine and its composition from major healthcare platforms '1mg' and 'WebMD' and provide users a concise yet reliable information to their request with ease, on a single platform.

2 Proposed Timeline

2.1 First Half

Employing web scraping to display information on medications and their ingredients based on user searches. (For example: Displaying summary of 'Pan-40' medicine based on user search)

Relevant information to be shown: Medicine usage, potential side effects and its composition.
Website to be used for scraping: **Tata 1mg (https://www.1mg.com/)**

2.2 Second Half

Employing web scraping to display summary of medicine ingredient (For example: Summary of the key ingredient of 'Pan-40' medication i.e. pantoprazole).

Relevant information to be shown: summary of the medicine's composition.
Website to be used for scraping: **WebMD (https://www.webmd.com/drugs/2/index)**

3 Installation and User Guidelines

To install the package, we need to clone the repository as follows. To clone via terminal

```
clone https://github.com/parismita/Drugpedia.git
```

Or we can directly download the zip package from github project page. After cloning, to install the application, goto the cloned folder Drugpedia and run the following command in terminal to install the python package.

```
make setup
```

Note: The dependencies are mentioned in requirement.txt file. With this the installation for python package is complete.

To run the application just run the command *make* in terminal inside the Drugpedia directory. This will start the development server at

```
http://localhost:8080
```

Now, we have to goto the browser and open the main page which is hosted on the above address.
Note: with this the website will fetch the data via web scraping and this data will not be stored in postgres database

To use the database feature which will store and retrieve the data from local postgreSQL, we need to install postgreSQL.
Database creation is as follows using psql

```
CREATE DATABASE drugpedia ;
```

Here I am creating the database named drugpedia, we can create the database with any name. To connect to the database we need to include these into the .env file where my user is postgres which is the default user created by postgreSQL, the database is drugpedia and my postgres is running in localhost at port 5432

```
USER=postgres
HOST=localhost
PASS=yourpassword
DB=drugpedia
POST=5432
```

Note: we need to create the .env file inside the project directory. After connecting to the database, we need to create the tables which can be done by using this api in our browser.

```
localhost :8080/ create
```

This will automatically create the required tables.

Now we are done with all installations and ready to use the website. Examples to use the APIs for searching for medicine and ingredients is given as follows

```
http :// localhost :8080/ medicine / search−json ?key=pan
http :// localhost :8080/ ingredient / search−json ?key=cro
```

Example to use the API endpoints for detail page

```
http :// localhost :8080/ medicine / details −json
http :// localhost :8080/ ingredient / details −json
```

With query params id and name which corresponds to the link and name from search results for both APIs. These will give us JSON response and if we want to browse through the website we can do it at

```
localhost :8080
```

4 Code Structure

Here's the higher level view of the file structure, for more detailed view please check on this link

- src - has the backend and frontend code

    - controllers - Rendering the pages and re-directions are written here.

        * homecontroller.py - render the main page
        * ingcontroller.py - controller for ingredient module
        * medcontroller.py - controller for medicine module

    - models - the SQLAlchemy models for the tables

        * medicine.py - The medicine table's model
        * ingredient.py - The ingredient table's model

    - services - main logic is written here

        * medservice.py - modules for CRUD, search and details of medicine
        * ingservice.py - modules for CRUD, search and details of ingredient

    - routes - routing on given urls

        * homeroute.py - main page route
        * ingroute.py - ingredient module routes
        * medroute.py - medicine module routes

    - utils - utility files

        * initdb.py - initializing the database here

    - template - contain the html codes

        * search-ingredient.html
        * search-ingredient-results.html
        * ingredient-details.html
        * index.html
        * search-results.html
        * medicine-details.html

- test - For unit testing

    - test.py - For unit testing, currently no tests there

- main.py - Creating the application, connecting to postgres here.

- Makefile - Build and Run the application, linting and installation of dependencies

- requirement.txt - The dependencies

- documentation - Contains this manual

5 Some Important Functions

- MedSearch(key) - the module for searching the medicine names, it takes the user input value to be searched as argument and returns the dict of medicine list. It calls two seperate search functions to do webscraping from otc and drugs lists and combine the results into one dataframe.

- MedDetails(id, name) - the module for the medicine summary, it takes the unique url and name to medicine as argument and returns the dict of medicine data. The details for otc and drugs are computed seperately and combined in this function.

- OtcDetails(content) - takes the parsed html as input and returns dict of otc summary

- DrugDetails(content) - takes the parsed html as input and returns dict of drugs summary

- IngredientDetails(id, name) - the module for the medicine summary, it takes the unique url and name to medicine as argument and returns the dict of ingredient data.

- IngredientSearch(key) - the module for searching the medicine names, it takes the user input value to be searched as argument and returns the dict of ingredient list.

- $create_db() - creates the postgreSQL tables$

6 Stack Used

The languages used are Python, HTML and Makefile. The dependencies for python are mentioned in requirement.txt file. For storing the data we used PostgreSQL. Version control is done via Github.

7 Results

7.1 UI for Medicine search

The front-end of our project has been successfully implemented using HTML, Internal CSS and Bootstrap 5.2 front-end toolkit. Users can search for a medicine by brand on the homepage of the website (See Fig. 1), or they can navigate to the ingredient search page, where they can search for a medicine's ingredient by name (See Fig. 2).

Once the user searches for a particular medicine, we return the number of medicines that match the user query, scraped from 1mg. This list is visible to the user after he is redirected to the medicine search results page. Fig. 3 showcases the results obtained post scraping for the user search of *'crocin'*. Now, the user
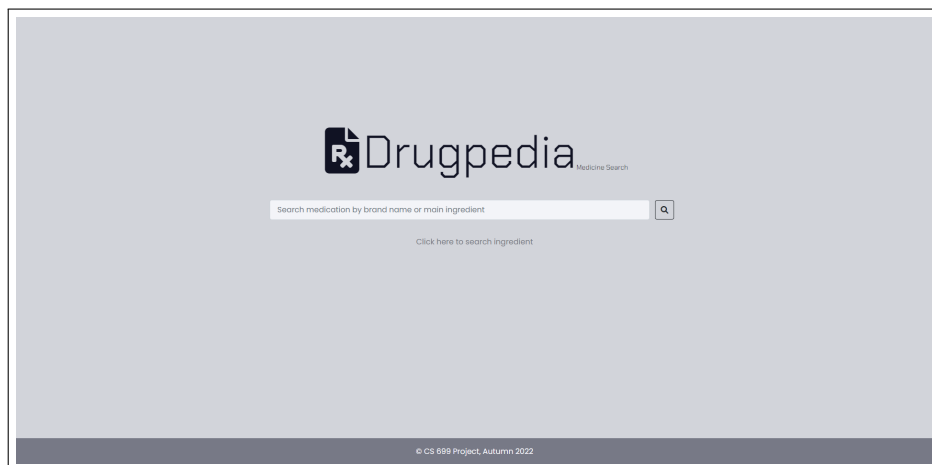
Figure 1: Landing page for medicine search, where a user can search for a medicine by brand or main ingredient.
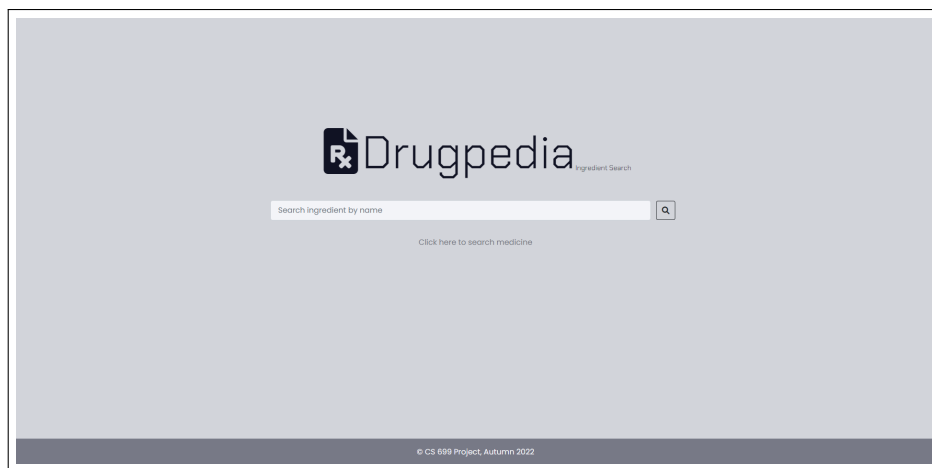

Figure 2: Landing page for ingredient search, where a user can search for a medicine's ingredient by name.
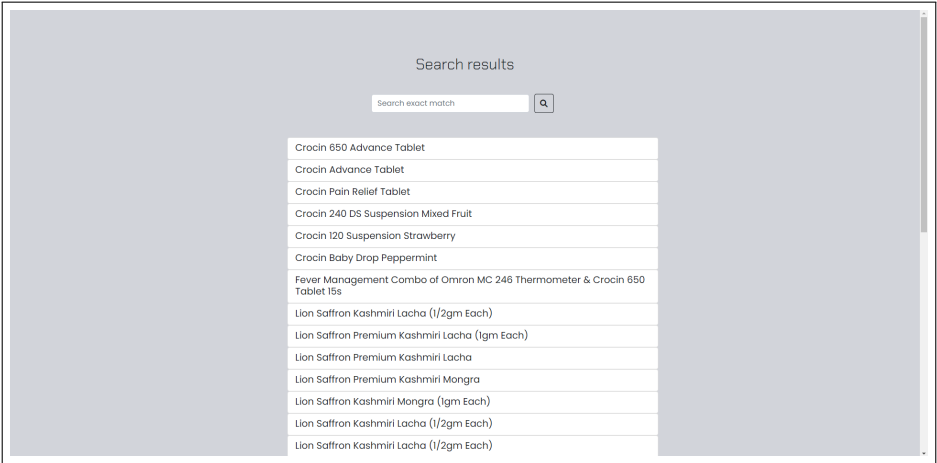
Figure 3: Search results page, showing the number of matches found on 1mg for a particular medicine searched by the user.

Figure 4: Medicine details page, showing the details of the exact matched medicine retrieved from 1mg.

can select the medicine which matches by exact name, and search for its details. Fig 4. shows the detailed view of a exact name-matched medicine (*'PAN 40 Tablet'*). Medicine's description, composition and side effects are scraped from 1mg and returned on the medicine details page.

7.2   UI for ingredient search

A similar flow was incorporated to return uses, side effects and precautions related to a medicine's ingredient (*'paracetamol'*), scraped from WebMD. (See Fig. 5 and Fig. 6).
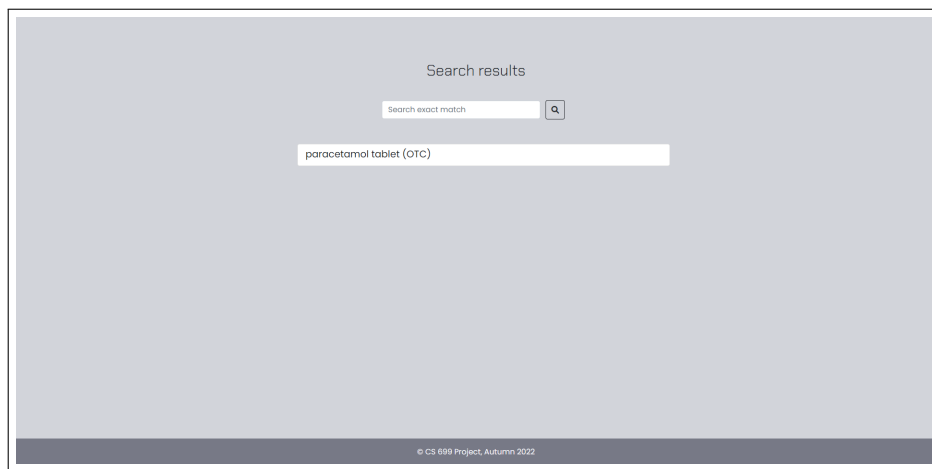

Figure 5: Search results page, showing the number of matches found on WebMD for a particular ingredient searched by the user.
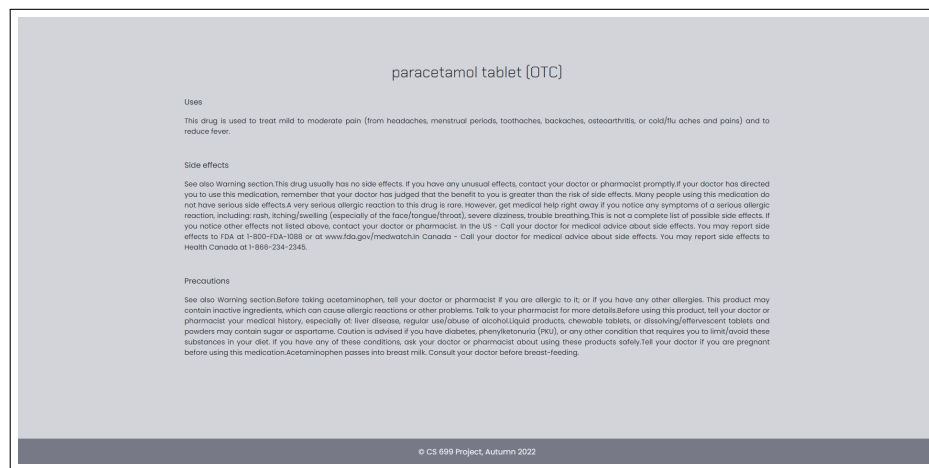
Figure 6: Ingredient details page, showing the details of the exact matched ingredient retrieved from WebMD.

8 Future Scope

- Incorporating scraped data from region specific medicine encyclopedias to include information of medicine brands specific to the region. E.g. `Drugs.com` provides drug information for medicines primarily available in the US.

- Incorporating scraped data from **'WHODrug Global'**. WHODrug data includes herbal remedies too, along with conventional medicine information.

9 Conclusion

Drugpedia runs on the browser, making it easily accessible. Scraping and displaying only relevant information of a medicine will help users to reduce browsing time.

References

[1] Edward Livingston Trudeau, To Comfort Always, Independently Published, 2016, ISBN 1519051077, 9781519051073

[2] Python3 library documentation. https://www.python.org/doc/

[3] BeautifulSoup documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[4] SQLAlchemy documentation. https://docs.sqlalchemy.org/en/14/

[5] Flask documentation. https://flask.palletsprojects.com/en/2.2.x/

[6] Jinga documentation. https://jinja.palletsprojects.com/en/3.1.x/