SENG3011 Deliverable 1 - Initial Design Report

Group: FatDonkey

Members: Flynn Zhang, Edwin Tang, Frank Su, Josh Rozario

About Us

Due to unprecedented events from the past year, our team has been tasked by ISER (Integrated Systems for Epidemic Response) to develop an outbreak surveillance system to further assist their Epiwatch software which uses public data sources to detect outbreaks.

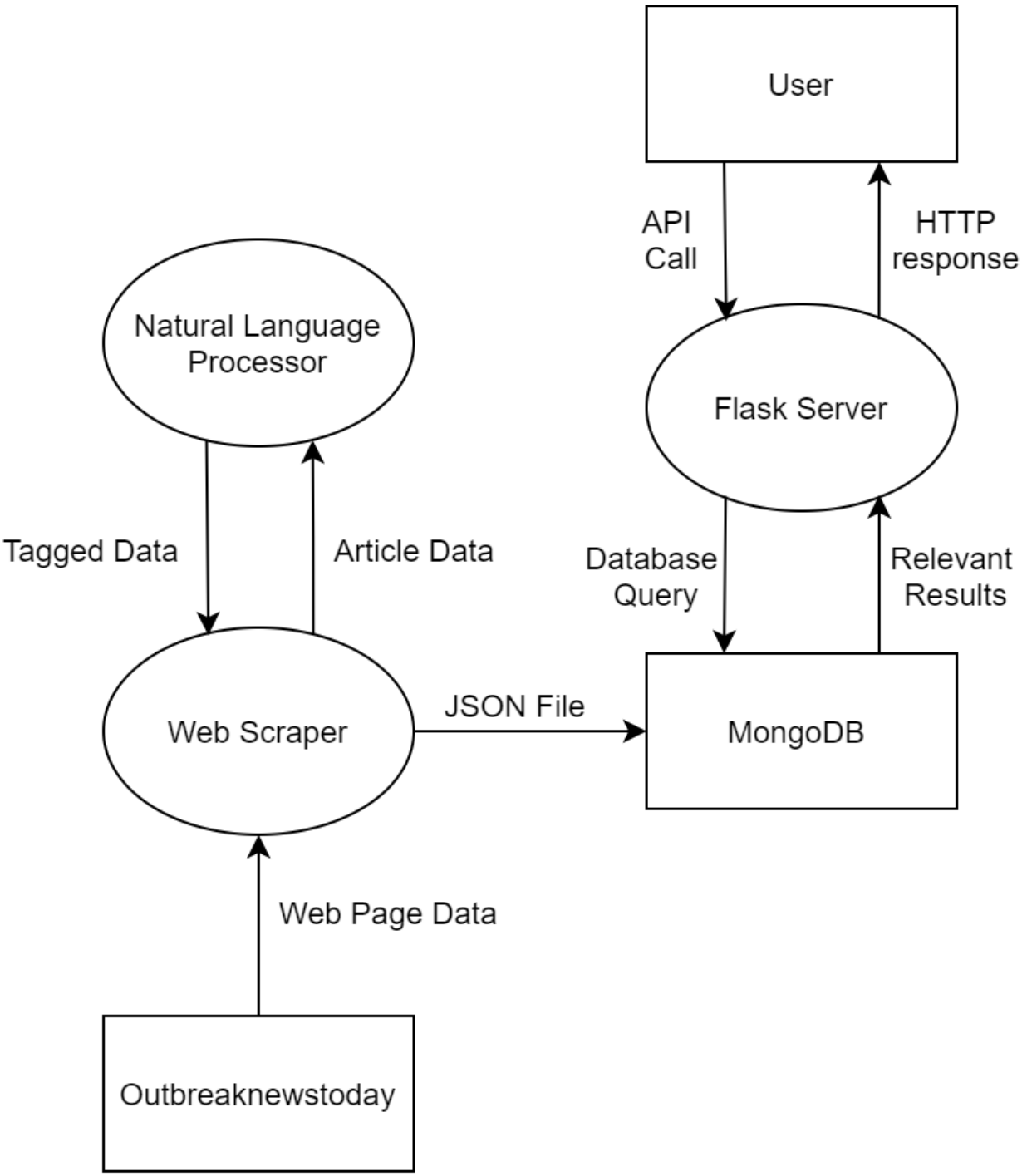
The following report details stage 1 of our project which is to develop an API which retrieves disease reports from a data source (outbreaknewstoday.com) that are relevant to the parameters parsed to it. Our rationale for programming language, hosting, storage and framework decisions will be included.

In stage 2 of our project, we will be developing a web application which allows users to interact with our API in a user friendly way.

System Overview

Initially, the articles from the site will be recursively scraped and stored. After, all articles will be passed to a natural language processor to extract the name of the disease and the location of the disease and add that information to each article element. The tagged data will then be stored in a database for easy retrieval from our API. A user will be able to call our web server after it has been deployed to request their desired data. In order to maintain an updated database, we will scrape and tag articles once a day.

The following diagram is a visual depiction of how our system will operate:


User
API
Call
HTTP
response
Natural Language
Processor
Flask Server
Tagged Data
Article Data
Database
Query
Relevant
Results
Web Scraper
JSON File
MongoDB
Web Page Data
Outbreaknewstoday

API Module

We intend to use Python and Flask to develop our Restful API and scraper as we have all used the framework and language in the past.

The scraping of our data source http://outbreaknewstoday.com/ will be carried out using Scrapy, a popular Python module that our team also has experience with. It will extract the date, and title of the article as well as the body of news reports. Extracting the metadata will help categorise the reports and help users of our service refine their search. We believe there will be certain challenges in extracting the location and disease name metadata as this information is embedded within the title of the news report. To solve this problem, we will use text analysis AI (Spacy) to extract the desired data. The data will be stored as json files and imported into a MongoDB database. The API will be able to efficiently query this database without having to access each individual element inside a json file whenever a call is executed. After observing the frequency of articles published to the website, we believe that scraping the data daily will be sufficient for maintaining an up-to-date API.

As mentioned previously, we will use MongoDB as our NoSQL database management system which is perfect for processing our document based data (json files). Our API will be built using Flask, a lightweight web server framework. Calls to our API will be received and handled by our Flask server which will fetch the relevant data from the database. The reply will also be packaged as a JSON file. We will use the Flask-Restplus extension to automatically document our API endpoints through a Swagger UI.

We plan to deploy the API to the web using Heroku's Cloud application platform for its convenience and widespread availability.

Module Interaction

We will be using our learnt REST principles and the HTTP protocol to pass parameters to our modules. We are using REST as it is the de facto standard for creating the architecture of network systems. Furthermore, we are also using HTTP which a lot of our members are familiar with as they have used it in earlier courses. We will first start with implementing the most common HTTP methods, GET, PUT, DELETE and POST. With the current specification we have been provided, we will most likely be using mainly the GET method. In the GET method the user is to pass certain parameters that we would use to then return the most relevant items. Some of the parameters would most likely be the location of the article, the strain of the disease and the date of the article.

Our module would then find the relevant articles that matches the user's parameters from the database. The API call will then be processed and will return to the user, a list of the relevant articles as a JSON object.

The object returned would most likely be a JSON object. This is because most web-based systems implement a Javascript based framework, thus allowing our API to be more compatible with more users and increasing our user market.

If there is an error during the execution of the API request, it will be handled as per HTTP response codes. E.g. 404 - Web address does not exist, 405 - Method not allowed, 500 - Internal Server Error - API is not online.

API Call Examples

Get /www.ourapi.com/?location=Sydney&date=11122000&disease=Swine
Accept: text/html
Accept=Language: en-us, en

Get /www.ourapi.com/?location=USA&date=11112020&disease=covid
Accept: text/html
Accept=Language: en-us, en

JSON Return Example

```
{
    “article”: {
        “metadata” : {
            “location” : “sydney”,
            “disease” : “covid”,
            “date” : 11122000”
```

},

“text” : “The US is at risk of losing all its recent gains in the battle against COVID-19 as highly contagious variants take advantage of Americans getting lax with safety measures.

Please hear me clearly: At this level of cases with variants spreading, we stand to completely lose the hard earned ground we have gained," said Dr Rochelle Walensky, director of the US Centres for Disease Control and Prevention.

After weeks of tumbling case numbers, new infections are on the rise again - about two per cent more this past week compared to the previous week, Dr Walensky said on Monday.”

},

etc..

}

Implementation Decisions and Rationale

a. Python
    i. All group members have familiarity with Python and so we can minimise the amount of time spent learning the language
    ii. Language is suitable for the scope and agile development of our project
    iii. Existing libraries will streamline development of our program as various functionalities (e.g. Natural language processing) can be easily provided by Python libraries
    iv. De-facto programming language to interact with the web.
b. Flask
    i. All group members have familiarity with this library from previous projects.
    ii. Serves as easy way to process API calls through implementing them as HTTP endpoints
c. Flask-restplus
    i. Python library to automatically document our API as we develop it by providing a Swagger UI.
d. MongoDB
    i. Simple and effective NoSQL database management system but is still powerful that will allow us to easily store the scraped data.
    ii. Can be easily set up to interact with Python through Python scripts
    iii. NoSQL can provide superior performance in comparison to relation DBMS
    iv. Does not require data to be transformed into a schema set by the database
    v. Existing data stored in JSON file can easily be imported into a MongoDB database
e. PyMongo
    i. Python library to allow interactions with the MongoDB database
f. CSE Vlab
    i. Default development environment for this project as many of the tools going to be used are already available
    ii. Using CSE Vlab ensures consistency in the development environment among all group members
g. Scrapy.py
    i. Python library for web scraping that will allow us to scrape data from our allocated data source
    ii. Familiarity with this library from previous projects will streamline the development of our scraping program
h. Heroku
    i. Our application is planned to be deployed on Heroku as it is more suitable when considering our project scope in comparison to AWS.

    ii. Allows for easy and flexible management of our system
    iii. Uses Ubuntu 20.04 (default version) as the operating system for the web server which will be supported in the long-term (2025)

i. Spacy
    i. Python libraries to help us easily extract the disease names and locations from the title of each data source
    ii. Can experiment with a variety of prediction models to find one that is the best at identifying desired metadata

Challenges

Classifying articles/Data processing

Extracting the metadata from articles is an important task as we believe having a single field to identify the location and disease mentioned in the article is critical for fast database queries. Performing this task has been quite challenging as the models used are not very good identifying disease names as diseases may be abbreviated (e.g. H9N2) or may have different forms (e.g. Coronavirus vs Covid-19).

We have had more success with identifying locations as the models are more familiar with geographical names but there are also difficulties involved. Locations that are identified can be cities, states, countries or even continents which creates inconsistencies in the type of value that the field may hold. This could result in queries missing certain articles that could be relevant to the search.

Another issue related to processing the raw data from the site, is that the body of the article may mention more than 1 location or disease. Adding these values to the metadata of an article would slow down the database query as the database would need to carry out multiple comparison operations in the event of a query. The complexity of this kind of search would be O(**m** x **n**) where **m** is the average number of tags per article and **n** is the number of articles. In comparison, if an article only had 1 value per article the complexity would be O(n).

High Coupling

The API is over reliant on the functioning of the scraper and its consequent text analyser to perform its role accurately in order to function. Furthermore, the flask server is reliant on the correct functioning of our database so it can accurately return the correct information to the user. Evidently, each component of our system is reliant on another and a malfunction of a single component may have a domino effect on the rest of the system and will affect the endpoint.

This problem is difficult to address due to the nature of the system. Since, our system only consists of 3-4 components (web server, database, natural language processor), we believe this is an acceptable design flaw since we have the necessary manpower and time to ensure that each component performs as intended. This problem may become more pronounced if our system is scaled up to include more components in the future.