SENG 3011 -

Design Details

“Weekly Cri Sesh”

Madeline Dobbie (z5112961), Nathan Driscoll (z5204935), Shabrina Yusri (z5205391), Sisil Harry Gunasekera (z5113599), Sumeet Charan (z5160224).

Table of Contents

Introduction **2**

Design Details **3**

Development of API 3

Structure 3

Web Service Mode 4

API Interaction 5

API Specifications 5

Possible Interaction Example 5

Data Collection and Processing 7

Tools and Libraries 9

Implementation Language(s) 9

Development environment 9

Deployment environment 9

Specific Libraries 9

Introduction

Before 2020, the concepts of wearing a face mask everywhere you go, seeing state borders locking down and being restricted to not seeing your family and friends indefinitely were unimaginable for most and even apocalyptic to some. Yet, after the year had passed, this was the world society had to live in due to the COVID-19 outbreak that infected the globe.

One of the leading voices in the battle to suppress and control COVID-19 is the World Health Organisation (WHO), which monitors potential disease outbreaks through their comprehensive database of reports from their members across the globe. As seen, the management of COVID-19's spread was poor, partly due to organisations and governments not having easy and quick access to the most accurate information from this database.

Due to this, our team is aiming to develop an API that can access these disease reports from WHO and other sources. It will enable the creation of applications that can identify and analyse potential outbreaks for users that can manage it before it sparks out of control. Through the applications, users can simply input key terms like "COVID-19" to get the most relevant and reliable information about diseases to help prevent possible epidemics.

Reports and Design Details

Development of API

Describe how you intend to develop the API module and provide the ability to run it in Web service mode.

Our team has chosen to use the agile development method in constructing our API. This will allow for changing definitions in requirements as the project develops. It will also encourage individual components of the project to have low dependency on others as they are created separately, leading to fewer errors. Through this we will also aim to follow the SOLID principles (Single Responsibility, Dependency Inversion and Open Closed, etc) to allow the API to be highly integratable with applications.

Structure

As this API will interact with many sources of information as well as the applications it enables, there will be many levels of interaction. This leads to a necessity for clear structure to avoid coupling and high dependency (Fig 1). This structure ensures that ‘levels’ only interact with a level directly below or above it.

This ensures low dependency and a clear common method of interaction between them, so that this API can be highly integratable with many sources and applications. It also improves security, ensuring that users can not get direct access to the certain servers as authentication is required before passing to that layer. Furthermore, it assists in separating components so that team members can work on individual components without affecting others' work.

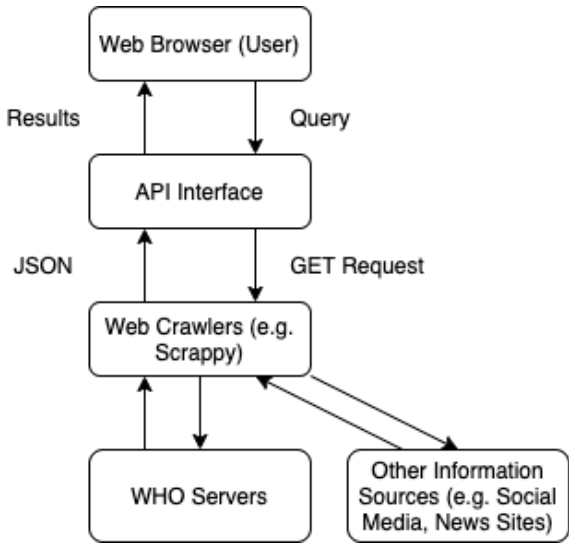

Fig 1: Structure of API interactions.

Web Service Mode

We will provide the ability to run in web service mode by utilising Flask. Flask will be set up to process API requests as well as a server created to run the API. Once active, the API will manage GET requests to request information. This will be returned in JSON encoded responses that will be interpreted before returning to the user (Fig 2). The get requests will take the users search term into the GET Request for further processing.

Our team has decided to use Flask, as we are highly experienced in it through university projects and have experience in using it in conjunction with APIs.

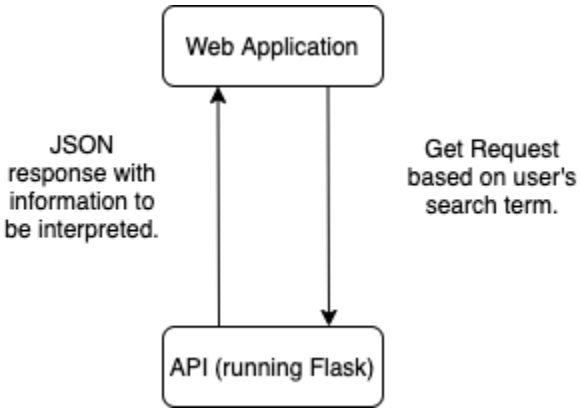

Fig 2: Interaction via Web Service Mode

API Interaction

Discuss your current thinking about how parameters can be passed to your module and how results are collected. Show an example of a possible interaction (e.g.- sample HTTP calls with URL and parameters).

API Specifications

Our users will utilize the API through a website, which they will use to input their desired parameters into the respective fields. The form generated will then be restructured into a GET request made to the API. The parameters will be carried in the body of the request rather than in the URL to allow for a more extensive range of parameters to be taken and to prevent the URL from becoming excessively long or cluttered.

Responses will be returned in the commonly and often used format of JSON, containing a list of articles, each article referring to a single news item or report from WHO. Each article will return basic information taken from the resource, such as disease-name, dates, locations, number-affected and differentiation of victims, e.g. "death", "infected", "hospitalised", “recovered".

Possible Interaction Example

Interaction will be done as shown below (Fig 3, Fig 4).

```
curl -X GET https://weekly_cri_sesh_api_url
-H 'Content-Type: application/json'
-d '{
   "dateStart":"2020-08-01T00:00:00.00",
   "dateEnd":"2021-03-04T00:00:00.00",
   "diseaseName":"Swine Flu",
   "location":"China"
}'
```

Fig 3: Get Request containing users search terms.

```
{
    "articles":[
        {
            "url":"https://www.who.int/news/item/13-06-2020-a-cluste
            r-of-covid-19-in-beijing-people-s-republic-of-china",
            "headline":"A cluster of COVID-19 in Beijing, People's
            Republic of China",
            "articleDate":"13 June 2020"
            "reports": [
                {
                    "diseases": [
                        "COVID-19"
                    ],
                    "eventDateStart":"",
                    "eventDateEnd":"2020-06-13T00:00:00.00",
                    "locations": [
                        "Beijing",
                        "China",
                        "Xinfadi"
                    ],
                    "symptoms": [
                        "fever"
                    ],
                    "numberAffected": [
                        "recovered":"",
                        "dead":"",
                        "hospitalised":"",
                        "infected":"87",
                        "symptomatic":"41",
                        "asymptomatic":"46"
                    ]
                }
            ]
        }
        {
            "url":"https://www.who.int/news/item/28-01-2020-who-chin
            a-leaders-discuss-next-steps-in-battle-against-coronavir
            us-outbreak",
```


Fig 4: JSON Response from Get Request

Data Collection and Processing

Upon receiving a GET request from our frontend web application, the API will initially check the SQLite database of cached reports and check if the reports cached are a superset of the new GET request. If the GET request filter parameters can be satisfied with already cached data, then the API will return the data in the format as detailed in the API Specifications section.

However, if the GET request cannot be satisfied with the already cached data, then the API will direct the crawler to scrape through related articles on WHO's website chronologically. The crawler will obtain data that is relevant to the filter parameters passed through the GET request. The obtained data will be processed by the crawler into a JSON containing a list of articles, each article will contain information specific to a single article and any related data. This data will get passed from the crawler to the API which will return it to the frontend web application to be displayed in a readable and user friendly manner.

Status codes will be returned by the API under these conditions (Fig 5).

Status Code		Description
200	OK	Successful. We will use this code for all successful GET requests.
400	Bad Request	Bad input parameter. If there is an error in the GET request submitted, we will return this error code.
404	Not Found	Resource not found. WHO down and no cached data matching query parameters.
405	Method Not Allowed	The resource does not support the specified HTTP verb. As our API exclusively accepts GET requests. Any requests not using a GET request will be returned with this error. POST, PUT, PATCH and DELETE will return 405.
418	I'm a Teapot	The server refuses to attempt to brew coffee with a teapot
500	Internal Server Error	Server not working as expected, request is valid. If our server has any internal errors this error code will be returned.

Fig 5: Status codes API will use.

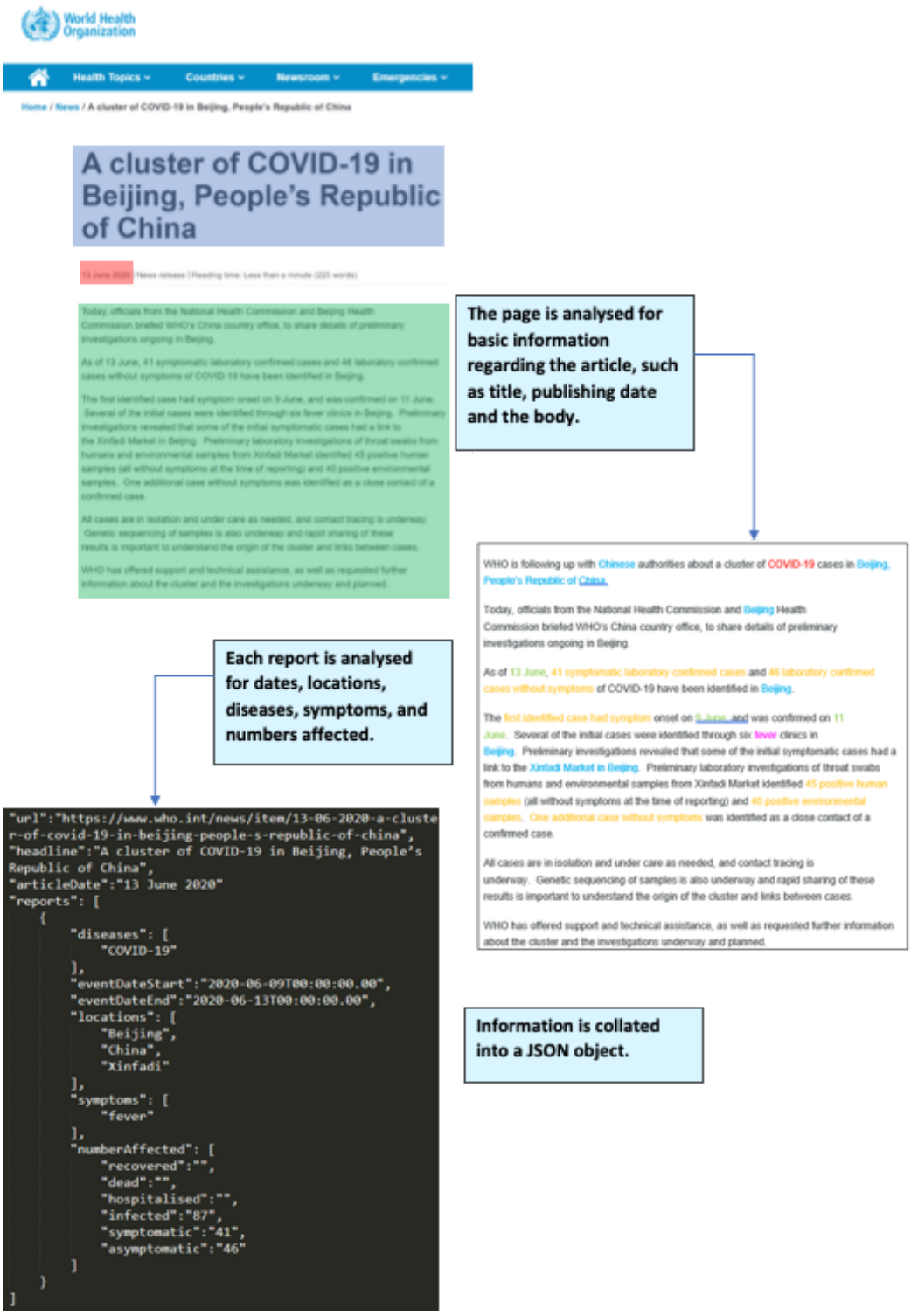

Fig 6: Processing of Data from all levels.

Tools and Libraries

Present and justify implementation language, development and deployment environment (e.g. Linux, Windows) and specific libraries that you plan to use.

Implementation Language(s)

Our group will mainly use Python for implementation. We chose Python as all team members are familiar with using Python having used it in similar projects before. By choosing Python, we can also use frameworks such as Flask and Scrapy which are built in Python.

Flask will be used for web functionality. Flask was chosen as it is popular, simple to use, and we have used it before as a group.

For the frontend, our team will use HTML, CSS and JS. These tools will be used because they are commonly used for frontend development and are simple to use.

Development environment

Our group will use GitHub as a code repository. GitHub uses Git which offers version control which allows team members to revert code to a previous version in the case that the current code has errors. GitHub allows all team members to collaboratively contribute to the project across the internet.

We will use text editors such as Visual Studio Code (VSC) and Sublime to develop the project. These text editors have many helpful features including GitHub integration which make development easier.

Deployment environment

We will use Stoplight.io for our API. Stoplight's free plan allows us to design, develop and document our API. The reason we chose Stoplight over a similar API development tool such as Swagger was that Spotlight's free plan allowed 5 users to collaborate, whereas Swagger only allowed 1 user. Using Stoplight will allow all members of the group to develop the API.

For our server we will use Ubuntu as it is very simple to set up a server in Ubuntu. Our group is comfortable with using Ubuntu for a server as we have used it for a server in the past for previous projects. There is also plentiful support online for an Ubuntu server as it is widely used.

Specific Libraries

Scrapy is an application framework designed for web scraping. We will need Scrapy to scrape the WHO website for medical reports. Scrapy was chosen as it is free, open source and used with Python (our main implementation language). It also has comprehensive documentation and tutorials.

SQLite is the database software we will use. The database will be used to store cached terms and reports from Scrapy scraping the WHO website. SQLite was chosen as it is a widely used DBMS that our group has had some experience with in the past. Since this is a relatively small project, we felt that it was unnecessary to host a separate server for a PostgreSQL database, and SQLite would streamline the process of connecting the database to our API. Also, we used SQLite in a previous similar project, and this worked well enough for its purposes.