DESIGN DETAILS

SENG3011 REPORT

TELEUBBIES
Yin Huey Tan z5211373
Amanda (Xiaorui) Li z5206613
Sarah Oakman z5206178
Lavanya Sood z5208121
Yiyun Yang z5187469

INTRODUCTION

The **EpiWATCH system** utilises multiple accessible data sources to automate epidemic detection. We are creating a REST API module to contribute to the **EpiWATCH system** by identifying disease reports from the WHO data source and in the future, developing a web platform to bring epidemic detection to end users.

The design phase is necessary in projects to discuss how the whole system will be developed and deployed. Python, Flask, SQL and JavaScript will be used as the development languages and various tools including Swagger and Postman were considered for the API documentation and testing. We also explored ways it could be run in web service mode and all types of parameters that can be passed into the API module and how the results are collected.

1.0 API DEVELOPMENT AND IMPLEMENTATION

1.1 WEB SCRAPING

The first step for developing an API module is to translate information on a webpage into data we can search and sort through. In order to 'read' and interpret the data and articles on WHO which will later be stored in databases, we need to create a spider (web-crawler) through Scrapy which is able to view and save text on website pages, as well as traverse pages. Scrapy crawler works by making request to the URL (https://www.who.int/emergencies/diseases/news/en/), and looks for specific information (e.g. report content, dates, author etc) within the HTML file of the URL through CSS Selector (e.g. based off what the div classes are called). It then creates Python dictionaries with the extracted information, looks for links to other pages (as seen below with links to other viruses, reports and the next page of results) and continues the process through calling another request.

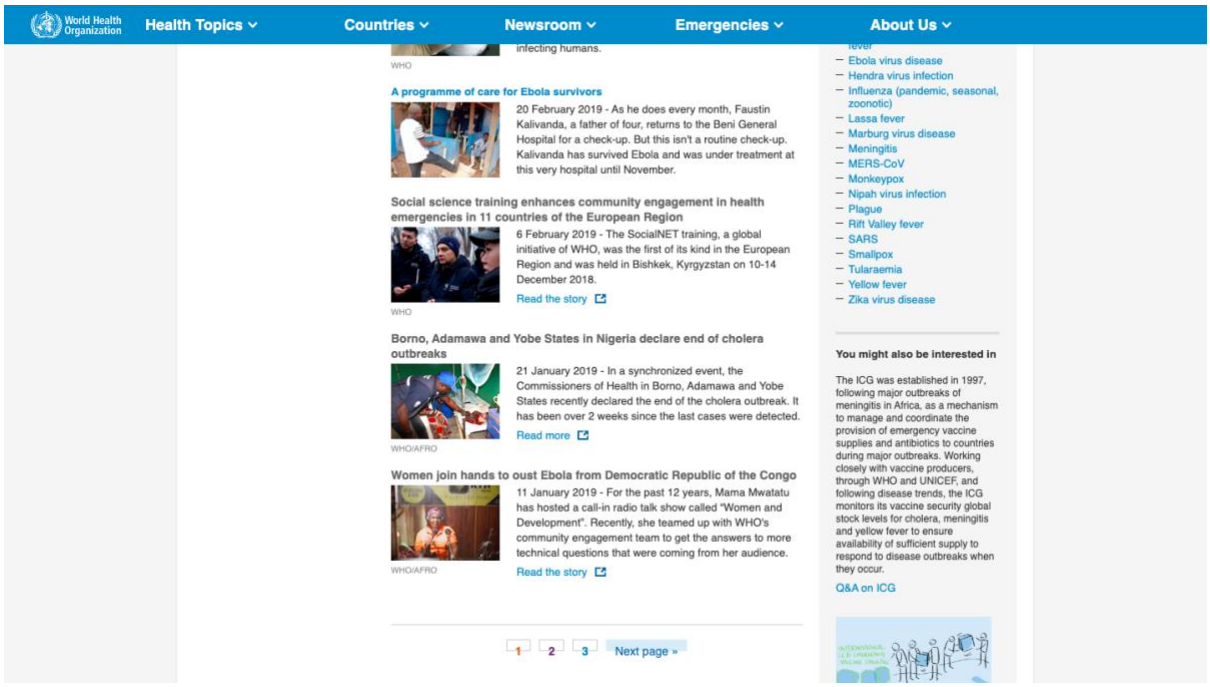

Fig 1.0 – Examples of the links available on the data source

If we wish to perform further operations (e.g. scanning through the dictionary and excluding certain data like reports we already have in our database), we can use 'Item Pipeline'.

Each item pipeline component (sometimes referred to as just "Item Pipeline") is a Python class that implements a simple method. They receive an item and perform an action over it, also deciding if the item should continue through the pipeline or be dropped and no longer processed. Item Pipelines can be used to clean HTML data, validate the scraped data (check items contain certain fields), checking for duplicates (and dropping them) and storing the item in a database.

After our data is validated and checked, we need to store it properly, which means generating an 'export file' (commonly called 'export feed') to be consumed by other systems. The dictionary data can then be stored using 'Feed Exports' which can generate either a JSON file following the JSON schema provided in specifications. From the JSON file created, we can load information into databases stored through SQLite (e.g. INSERT into <table>).

1.2 DATABASE

Another development aspect that needs to be considered for the API module is the database. The database structure is described below with all necessary fields outlined in the project specification and the WHO criteria included. The diagram outlines the relationships and keys between the tables and the fields present. Also, fields such as the PublicationDate, Location, Disease and Search Term will be indexed to allow for more efficient searching when parameters are given.

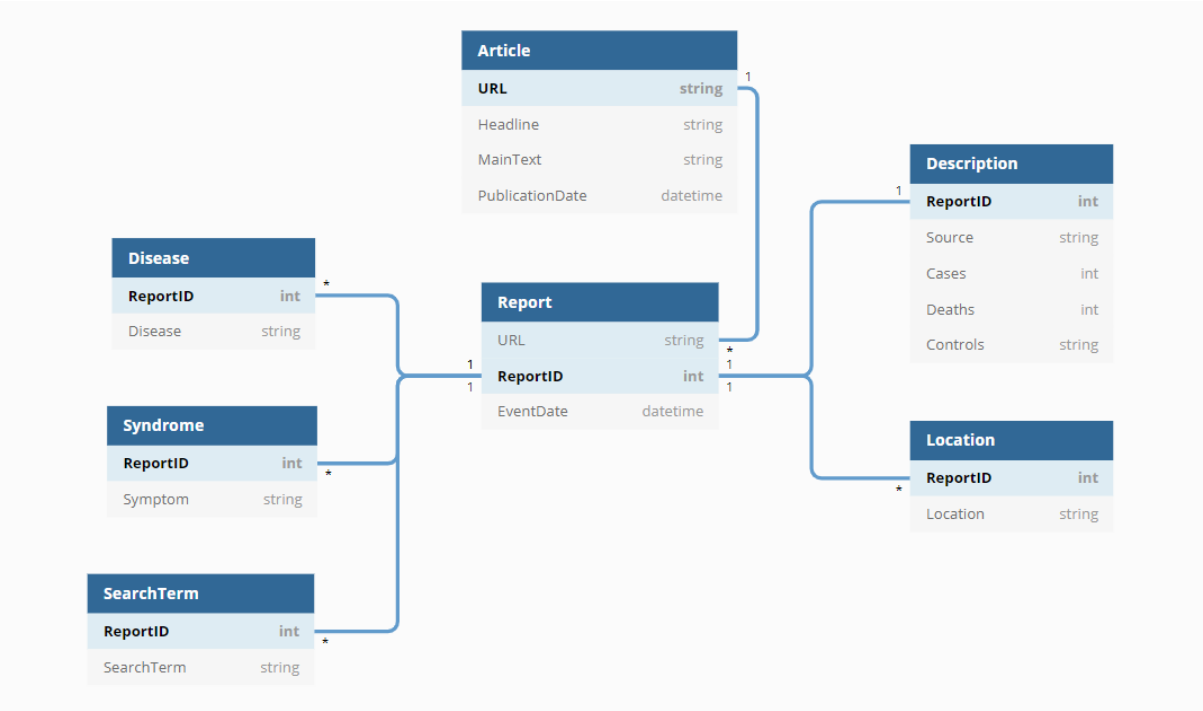

Fig 2.0 – The design and structure of the database

SQLite and SQLite Studio will be used to store, manage and show the information within the database. SQLite was selected since team members are familiar with the program, it can easily be imported and used within Python and it is compatible with several operating systems including Mac OS and Windows which is what the team will be using. SQLite Studio will be used to help visualise the database since it is built to work with SQLite and it is also compatible with the team's operating systems.

1.3 API DEVELOPMENT

To develop and build our API, we will be using Python with Flask since everyone in our team is familiar with this language and framework. We will be using Flask to map our HTTP requests to Python functions. The development of our API will be following the REST principle which is based on the four methods defined by the HTTP protocol : POST,GET,PUT and DELETE which correspond to the four traditional actions performed on data in a database: CREATE, READ, UPDATE, and DELETE.

In order to connect our API to our database, we will be importing sqlite3 library in python. An object representing the connection to the database is bound to the conn variable. We then create a cursor object which moves through the database to pull our data. Finally, we execute an SQL query with the cur.execute method to pull out all available data (*) from the table of our database. At the end of our function, the data is jsonified and returned in JSON format.

1.4 API TESTING

We will be using platforms like postman and swagger to test and validate our API request-responses at the development stage. This allows us to double check if our endpoints work as intended in different scenarios during the development process.

1.5 WEB SERVICE MODE AND API DOCUMENTATION

To run the API module in web service mode, Heroku and the Swagger User Interface will be utilised. Heroku is a suitable cloud based platform for delivering the API in web service mode as it can be linked to GitHub, supports Python and frameworks that are going to be used and it's free plan is suitable for our use.

Heroku will be used to deploy the API module to a server allowing it to be accessed on the Web. This will be achieved by linking the team's GitHub repository to the deployment method on Heroku and also within the API module. After providing a project name, Heroku supplies a URL which can be used to run the API in Web service mode.

The Swagger User Interface will then be used to document and show all of the requests that a user can make using the API module, such as a request to get all disease reports in a given date range. Since the API will be programmed in Python with Flask, to make it available on Swagger the 'flask_swagger_ui' library will be used, specifically the 'get_swaggerui_blueprint' module. To provide the API information and details with Swagger, a json document will be referenced by a static path entailing the API requests available. Once the Swagger blueprint is registered, it can be accessed and used to run and test the API module. This would provide users with proficient information to use the API module for disease outbreaks.

2.0 INTERACTIONS WITH THE API MODULE

2.1 API MODULE PARAMETERS

Parameters are passed to the module with the HTTP endpoints which will affect the API response by finding and presenting suitable disease reports and information. The parameters that the API will support are described in the table below. Query string parameters will be used so that users can input a period of interest, key terms, locations and time zones.

Query string parameter	Required/Optional	Description	Type
start_date	Required	The start date for the period of interest for disease reports.	String with format "yyyy-MM-ddTHH:mm:ss".
end_date	Required	The end date for the period of interest for disease reports.	String with format "yyyy-MM-ddTHH:mm:ss".
key_terms	Optional	If included, one or more key terms separated by commas. This retrieves disease reports with key terms found.	String (case insensitive and key terms separated by a comma if more than one are supplied e.g. ebola,coronavirus)
location	Optional	If included, one or more key terms separated by commas. This retrieves disease reports within the locations. A city, country, state or timezone can be given.	String (case insensitive and locations separated by a comma if more than one are supplied e.g. sydney,seoul)

2.2 API MODULE RESULTS

The results found by the module will be collected once a suitable HTTP call is made. Since the disease reports will be stored within tables in the database, the Python scraper will update and edit the database daily. This will improve the speed efficiency of the API since all or most of the data will already be stored and indexed. Once a HTTP call is made to the module, the Python scraper will check for and add any news reports that may have been added in between it's daily scraping. According to the parameters the user has inputted, the data will be fetched from the database using suitable SQL queries. If location parameters are given, the GeoDB API will be used to find any cities and/or countries within the parameters given, and all of the related locations will be used in the SQL queries to find all news reports within these areas. The results will then be converted to the JSON format in the structure detailed in the project specification. Finally, this API response will be documented in Swagger showing the response headers and body with appropriate status code responses. Due to the nature of REST API's, industry standard status code responses will be shown to inform users of the result of the response.

2.3 EXAMPLE HTTP CALLS

An interaction with the API module using a HTTP call generally has the structure of the API url with endpoints followed by the query parameters. Using the Heroku platform, this provides the ability to create a URL which will be as follows, 'https://teletubbies-WHO-api.herokuapp.com'. The endpoint that will be used to retrieve disease reports will simply be called 'report'. Then, the parameters will follow and multiple examples of different inputs have been displayed below with the start and end dates being 1/1/2020 at 9:00am 15 seconds and 25/2/2020 at 7:30pm 45 seconds.

Description	HTTP call
No optional parameters: all news between the start and end date are found with no filtering.	https://teletubbies-WHO-restapi.herokuapp.com/report?start_date=2020-01-01T09:00:15&end_date=2020-02-25T19:30:45&key_terms=&location=
A key term parameter: news between the start and end date filtered by the keyword.	https://teletubbies-WHO-restapi.herokuapp.com/report?start_date=2020-01-01T09:00:15&end_date=2020-02-25T19:30:45&key_terms=ebola&location=
A location parameter: news between the start and end date filtered by the location.	https://teletubbies-WHO-restapi.herokuapp.com/report?start_date=2020-01-01T09:00:15&end_date=2020-02-25T19:30:45&key_terms=&location=uganda
Multiple key terms: news between the start and end date filtered for any report that contains at least one of the key terms	https://teletubbies-WHO-restapi.herokuapp.com/report?start_date=2020-01-01T09:00:15&end_date=2020-02-25T19:30:45&key_terms=ebola,cholera,coronavirus&location=
Multiple locations, news between the start and end date filtered for any report that contains at least one of the locations	https://teletubbies-WHO-restapi.herokuapp.com/report?start_date=2020-01-01T09:00:15&end_date=2020-02-25T19:30:45&key_terms=&location=uganda,sydney,indonesia
Key term and location parameters: news between the start and end date filtered for any report with both the key term and location. The relationship can be further described as (keyterm1 OR keyterm2) AND (location1 OR location2)	https://teletubbies-WHO-restapi.herokuapp.com/report?start_date=2020-01-01T09:00:15&end_date=2020-02-25T19:30:45&key_terms=ebola&location=uganda https://teletubbies-WHO-restapi.herokuapp.com/report?start_date=2020-01-01T09:00:15&end_date=2020-02-25T19:30:45&key_terms=ebola,cholera&location=uganda,EDT

3.0 WEB APPLICATION IMPLEMENTATION

For the development of our application we are going to be using various implementation languages, development and deployment environment and specific libraries as shown in the web stack below:

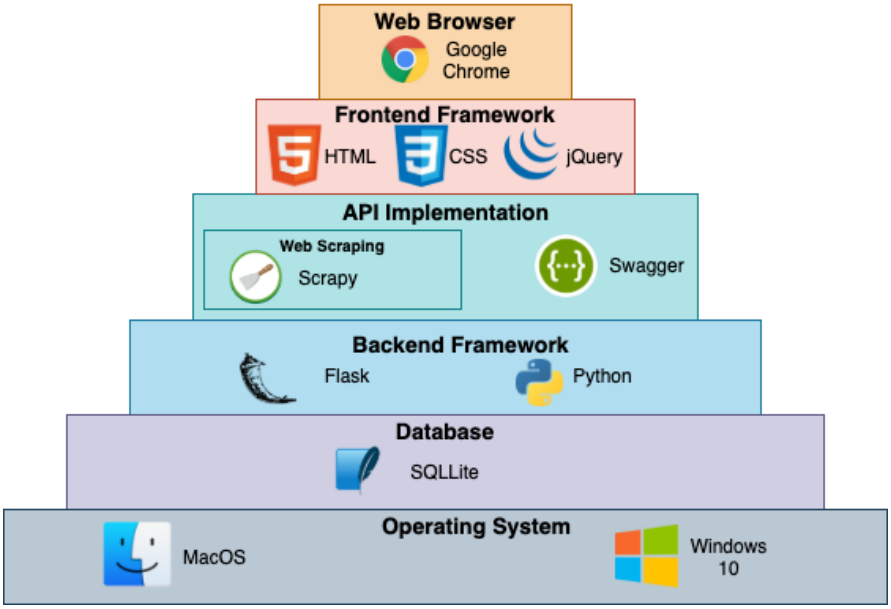

Fig 3.0 – The web stack

3.1 DEVELOPMENT AND DEPLOYMENT ENVIRONMENT

For the development of our web application we will be using the MacOSx Operating System and Windows 10 as they are the most frequently used operating systems. The programming languages such as Python and JavaScript which are going to be used for the implementation of the web application are fully compatible with both operating systems thus making it a good environment for development. Google Chrome web browser will also be used to access our web application. We will also be testing our application and our API across multiple platforms such as Windows, Linux etc throughout its development to test whether our application is working successfully.

3.2 DATABASE

SQLLite is going to be used for the database of the web application and is going to be used to store the information received from the API. SQLite is a very light weighted database so; it is easy to use it as an embedded software with multiple devices across different platforms. It has a very good performance and can be used to read and write data very quickly as it only loads the data which is needed, rather than reading the entire file and holding it in memory. SQLite queries are smaller than equivalent procedural codes so, chances of bugs are minimal. Thus, SQLLite is going to be used by the team for the web application database.

3.3 BACKEND FRAMEWORK AND API DEVELOPMENT

For the API development and the implementation of the backend framework we are going to be using Python and Flask. We are also going to be using SQLLite for the database and using Swagger and python libraries such as Scrapy for the development and maintaining of the API.

We are going to be using Python as it is a high-level object-oriented programming language with multiple resources available online. Python has good compatibility with Flask and is used to successfully build APIs. Python is very good at error handling and has good performance across multiple devices platforms. Due to the team members familiarity with the language through past experiences we believe that Python would be well suited for the development of the backend framework and the API.

Flask is a micro web framework written in Python which works as a webserver and allows in an easier redirection of web pages. Flask is easy to write, maintain and scale web applications and can allow users to successfully build APIs. Flask helps us to route requests to the appropriate handler and allows URLs to change without having to change the code. Due to flask being fully compatible and written it Python it can be used across multiple platforms and thus has been chosen by the team to be used for the development of the API and the backend framework.
Scrapy is a free and open-source web-crawling framework written in Python. It can be used to extract data using APIs or as a general-purpose web crawler. It is compatible across multiple devices and can run on various platforms. It processes and schedules the HTTP requests asynchronously and thus it can take care of multiple requests at a given time. This allows for other requests to continue even if there is an error that occurs while some request is being handled enabling us to make it suitable for API development for our project.

3.4 FRONTEND FRAMEWORK

For the implementation of the frontend framework we will be using JavaScript (jQuery), HTML and CSS.

We will be using HTML and CSS as they allow the information on the websites to be displayed in a systematic manner and allows us to develop our own layout for the website. CSS allows us to make our web application more aesthetically pleasing. HTML works cohesively with Flask which is going to be used in our backend implementation and thus making it a good resource for implementation.

We will be using JavaScript for frontend development as it allows the webpages to be more aesthetically pleasing and dynamic. JavaScript also has good compatibility across devices and works well on different operating systems and browsers. It can also quickly interpret and run the code which leads to less runtime errors. It supports multiple modules and there are multiple resources available to learn JavaScript from thus making it a good language for frontend development. As jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish and wraps them into methods that you can call with a single line of code, we will be using that library for our code implementation.