

Content

Developing the API Modules	2
Design and Implementation	2
Documentation	3
Testing	3
Authentication	3
Extension	3
Run it in Web service mode	3
API calls	4
Parameters	7
Calling the API	8
Examples:	8
Retrieving an article	8
Searching for articles	8
Software Architecture (API)	9
Programming Language	9
Development environment	9
Python libraries	9
Database	10
Deployment	10
Platform Design	11
Use Cases and Requirements	11
Design	18
Search	19
Results	19
Graph	19
Articles	20
News	21
Dashboard	22
Software Architecture (Application)	23
Programming Language	23
Development environment	23
Python libraries	23
Database	24
Deployment	24
API Integration	24
Generating Graphs	24

Other APIs used	25
EpiPro (WHO)	25
Google News	25

Developing the API Modules

Design and Implementation

We used REST (Representational State Transfer) architecture to build our API. REST is a style of architecture based on a set of principles (client/server, stateless, layered, and supports caching). In REST, the real world entity or an object is considered as a resource. Every resource is uniquely addressable using a uniform and minimal set of commands (typically using HTTP commands of GET, POST, PUT, or DELETE over the Internet). For this specific project, a resource is a report or an event. The architecture of REST is essentially the architecture of the Internet and that is why REST architecture is so popular.

Our API will serve as the service provider to provide disease reports to service consumers which are web applications or other APIs. The possible web applications that can be helped by our API include traveling applications, news applications etc. Basically, any application that needs to know about the risk of epidemic disease is our API's consumer.

For this project, we will have three independently working components, which are API, database and scraper. The scraper extracts data from the data source and stores the data in the database permanently twice a day. The API will get the data from the database and filter the data according to the clients' requests. Then, the API will create a dictionary as the response and send the response back to the clients in JSON format. This structure is decoupling and there are low dependencies between each component. So even if one component is broken or updated, the other components won't be affected much.

The inputs of the API are the period of interest (which is composed of "start_date" and "end_date"), "key_terms" and "location". The output of the API are disease reports satisfying those constraints.

Flask provides a route() decorator to handle the request for us. Then, we can use request.args function to get the parameters that the clients send to us. After getting all the parameters, we query the database using those parameters and extract data. In the end, we wrap the data in a dictionary, using the jsonify function to make a JSON file and sending the file back to the client.

Above is the flow of how every endpoint works. Since our API will only be retrieving articles and searching for articles, we came up with two endpoints.

/article/<article_id>

/articles

This is just our initial idea which may be changed during the process of implementation.

Documentation

Documentation is the key to a great experience when consuming our API. If we don't properly document our API, nobody is going to know how to use it, and it will be a horrible API. Good documentation is a concise reference manual containing all the information

required to work with the API, with details about the functions, classes, return types, arguments and more, supported by tutorials and examples. To publish our API in public, we plan to use Swagger. Swagger is an open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful Web services. Also, a framework called flask_restplus can automatically generate Swagger UI documentation for us. Since we are using flask, this framework is suitable for us to generate documentation.

Testing

To do testing, we will use the software called Postman. Postman can act as a client when testing the application developed in Restful state. It allows us to see the response of a particular endpoint that the server returns.

Before we deploy the API to servers, we will use our local host as the server to run the API.

Authentication

We only allow registered users to use our API, so we can monitor consumption of the API and know who did what. So we will store the user information such as username and password in the database and when we will check the password when the user tries to use our API. In flask_jwt library, there is a class called JWT allowing us to implement the authentication feature.

Extension

For the extension, we plan to generate statistic figures like graphs along with the reports. Of course the user can choose to receive them or not. For example, if the user search for the disease Zika from 11th March to 15th March, our API will generate a list of statistic figures according to the data stored in the database and send those figures back to the client along with the reports.

Perhaps statistic figures, graphs? Or some summary of outbreaks happened in a particular location, in a period of time? Notifications for registered users?

Run it in Web service mode

For web service, we need server to handle requests and send responses back, mostly in json format. For other APIs, they would request our service for data. We then access and retrieve data from database and finally return that data as json.

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Our api will be run in web service mode as it is:

Available over the internet since we are deploying it to DigitalOcean.

Uses a standardized JSON messaging system so every consumer can access and understand the response.

Is not tied to any one operating system or programming language.

Is discoverable via a simple find mechanism.

The web service platform we are using is HTTP + JSON. The data returned by the API is encoded as JSON (Content-Type: application/json). The JSON will follow the format that's given in the project specification. The character encoding is UTF-8. We might provide the response in XML format if there is a huge demand in the future (also as an extended feature).

HTTP status codes:

200 ---- Success

201 ---- Created

400 ---- Bad Request

404 ---- Not Found

500 ---- Internal Server Error

API calls

HTTP Verb	URL	Parameters	Response
GET	/article/<article_id>	article_id	200 Article found { "article_id": 3, "url": "http://www.cidrap.umn.edu/news-perspective/2019/03/long-us-flu-season-extends-shows-signs-decline", "date_of_publication": "2019-03-29T00:00:00", "headline": "Long US flu season extends but shows signs of decline", "main_text": "", "reports": [{ "report_id": 4, "disease": ["influenza"], "syndrome": ["Influenza-like illness"], "comment": "", "reported_events": [{ "event_id": 5, "type": "death, hospitalized, hospitalized", "date": "", "number_affected": 0,

			<pre> "location": { "location_id": 5, "location_name": "United States", "geonames_id": 0 } }] }] } </pre> <p>400 Invalid article ID</p> <p>404 Article not found</p> <p>500 Database error</p>
GET	/articles	start_date end_date key_terms location	200 Search successful <pre> { "articles": [{ "article_id": 2, "url": "http://www.cidrap.umn.edu/news-perspecti ve/2019/03/drc-ebola-total-grows-15-new-a ntiviral-clears-hurdle", "date_of_publication": "2019-03-29T00:00:00", "headline": "DRC Ebola total grows by 15; new antiviral clears hurdle", "main_text": "", "reports": [{ "report_id": 3, "disease": ["ebola"], "syndrome": [], "comment": "", "reported_events": [{ "event_id": 4, "type": "death, infected, infected", "date": "", "number_affected": 0, </pre>

			<pre> "location": { "location_id": 4, "location_name": "unknown", "geonames_id": 0 } }] }] }, { "article_id": 3, "url": "http://www.cidrap.umn.edu/news-perspecti ve/2019/03/long-us-flu-season-extends-sho ws-signs-decline", "date_of_publication": "2019-03-29T00:00:00", "headline": "Long US flu season extends but shows signs of decline", "main_text": "", "reports": [{ "report_id": 4, "disease": ["influenza"], "syndrome": ["Influenza-like illness"], "comment": "", "reported_events": [{ "event_id": 5, "type": "death, hospitalized, hospitalized", "date": "", "number_affected": 0, "location": { "location_id": 5, "location_name": "United States", "geonames_id": 0 } }] }] }] } </pre>
--	--	--	---

			400 Invalid parameters 404 No results found 500 Database error
--	--	--	---

Parameters

Parameter	Required	Type	Description
article_id	yes	Integer	ID of the article
start_date	yes	String in the form: “yyyy-MM-ddTHH:mm:ss”	Start date of articles in period of interest. Earliest start_date available is “2018-07-23T00:00:00”. Year is required, every other segment is optional, but missing characters must be replaced with “x”.
end_date	yes	String in the form: “yyyy-MM-ddTHH:mm:ss”	End date of articles in period of interest. Year is required, every other segment is optional, but missing characters must be replaced with "x". end_date must be identical to or chronologically after start_date.
key_terms	no	String containing search terms, separated by commas	Comma separated list of key terms to be search for
location	no	String	Name of location of interest

For GET /article/<article_id>, we passed article_id in as a path variable because it is used to retrieve only one resource (an article). It is clear that this number refers to the particular article retrieved.

However for GET /articles, we passed parameters as a query string instead. The main reason was that the key_terms and location are optional arguments - they may not be

provided, so these two could not be included as path variables. We considered having `start_date` and `end_date` as path variables since these were required arguments. The problem with this is that the meaning of these dates inside the path is ambiguous. For example, with the path `"/articles/2019-01-12Txx:xx:xx/2019-02-12Txx:xx:xx"` users may be unsure what these two dates mean. Using a query string instead brings clarity to the meanings of these dates:

`"/articles?start_date=2019-01-12Txx:xx:xx&end_date=2019-02-12Txx:xx:xx"`.

Calling the API

The API base URL is <http://www.doctorwhoseng.tk/>.

API calls are made by performing requests to the desired endpoint.
Data returned by the API is encoded as JSON.

Examples:

Retrieving an article

To retrieve the article with ID 5:

Make a GET request to <http://www.doctorwhoseng.tk/article/5>

The response is a JSON containing details of the article.

Searching for articles

To search for articles between 12 January 2019 and 12 February 2019, with key terms zika and influenza, and location Australia:

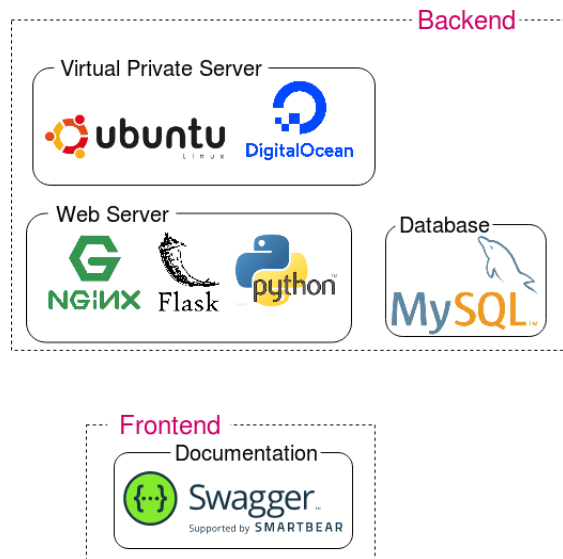
Make a GET request to

http://www.doctorwhoseng.tk/articles?start_date=2019-01-12Txx:xx:xx&end_date=2019-02-12Txx:xx:xx&key_terms=zika,influenza&location=Australia

The response is a JSON containing a field 'articles', which has a list of the articles that fulfil the search criteria as its value.

Software Architecture (API)

API



Programming Language

For this project, we will be using Python in conjunction with the Flask framework. The two main languages we considered for developing the API and website were Python and NodeJS. Although NodeJS offers good performance, we believed that Python was a better choice for our group to use.

The main reason we decided on Python over NodeJS was comfort. Not all of the members in our group had experience using NodeJS, however we were all familiar with using Python. Writing code in Python is much easier and faster than in NodeJS, because Python's compact syntax requires fewer lines of code to achieve the same result. In addition, the larger variety of Python libraries facilitate the development of our API and website. For example, the BeautifulSoup library was used for scraping the CIDRAP website to extract the information our API will be accessing.

Development environment

The next decision was whether to use Django or Flask framework. Django has a lot of built in features, making it easy to start developing. Flask is more lightweight, but allows for greater flexibility. Also, Flask has many useful libraries for building the API. For example, flask has a library called flask_restful that's designed for building RESTful api. This is why we chose Flask.

Python libraries

Since we are using flask framework, we are going to use those libraries to build the api: flask, flask_restful, flask_jwt, security, user

To make the scraper, we are going to use those libraries:
BeautifulSoup, requests, json

We also use other libraries to make new independent thread to update database daily such as:
threading, time, datetime

Database

We decide to use MySQL as our database for this project. The two options that we considered were MySQL and MongoDB. Although MongoDB stores data in JSON format and the output format of the API is defined as JSON format, MongoDB is more used in NodeJS programs while our team has decided to develop the backend in Python. Besides, since our team members all have experience using MySQL, it's easier for us to design and manage the database structure with it. We can store different aspects of the news such as time, location and disease type in the database by designing schemas for each of them. And we can then use commands like "SELECT", "UPDATE", "INSERT" and "DELETE" to query the data and manage it. To get the required output format, we can transform the filtered data into JSON files before outputting. Therefore, MySQL is a better choice for our project.

Deployment

After API implementation in backend, we decided to deploy the application on DigitalOcean Virtual Private Server running Linux system (Ubuntu 18.04) with all built-in dependencies required to run our application (Python, MySQL database, etc.). We can also register a domain name and have it point to the address of our VPS on DigitalOcean.

Reasons for choosing DigitalOcean VPS are:

- We get educational pack from Github for UNSW students credit
- Some of our members had experience running applications from DigitalOcean VPS
- DigitalOcean VPS provides many features, in which it can be run with dependencies and the environment of our choice, so we can run the application in the same environment that we implemented it in locally

Platform Design

Use Cases and Requirements

API requirements:

1. The API should be able to provide disease reports on demand, following the input and output formats defined
 - 1.1. The API should return disease reports according to the inputs:
 - 1.1.1 The API should return all the articles in the defined output format in the specified period of time when requested with a start date and end date
 - 1.1.2. The API should filter out the articles with the keyword within the specified period of time when requested with a start date, end date and a key term.
 - 1.1.3. The API should filter out the articles with the specified location within the specified period of time when requested with a start date, end date and a location.
 - 1.1.4. The API should filter out the articles with the keyword within the specified period of time first, then filter out the articles in the specified location when requested with a start date, end date, a key term and a location.
 - 1.2. The API should return disease reports in the defined format

Analytic Platform:

1. Users can search for the articles and news on the homepage related to their input
 - 1.1. Users can search for the articles and news within specific start date and end date
 - 1.1.1. The system will display all the news containing the keyword 'outbreak' during the inputted period of time
 - 1.2. Users can search for the articles and news within specific start date and end date containing a key term
 - 1.3. Users can search for the articles and news in a specific location within specific start date and end date
 - 1.3.1. The system will display all the news containing the keyword 'outbreak' in the specified location during the inputted period of time
 - 1.4. Users can search for the articles in a specific location within specific start date and end date containing a key term

2. Users can view a graph as an overview for the search results

2.1. Users can view the line chart as an overview of the results

2.1.1. Users can view a line chart displaying the number of reports containing the key term as disease on each day from the start date to the end date

2.1.2. Users can view a line chart displaying the number of reports containing and the key term as disease in the specified location on each day from the start date to the end date

2.2. Users can view the donut chart as an overview of the results

2.2.1. Users can view a donut chart displaying the distribution of the number of reports of different diseases

2.2.1. Users can view a donut chart displaying the distribution of the number of reports of different diseases in the specified location

3. Users can view the summary of reports containing the following:

3.1. Title

3.2. Date published

3.3. Users can click the 'More Info' button to view the following information:

i. Title

ii. URL

iii. Date published

iv. Main text

v. Reports

a. Disease

b. Type

c. Location

d. Number affected

4. Users can view the news containing the following information:

4.1. Title

4.2. Date Published

4.3. Users can click the 'More Info' button to view the following information:

i. Title

ii. URL

- iii. Author
- iv. Date published
- v. Description
- vi. A short version of the content

5. Users can create an account and log in to access the save and subscribe feature of the website

5.1. Users can create their personal accounts using the required information

- i. Name
- ii. Email
- iii. Username
- iv. Password
- v. Re-enter Password

5.2. Users can log into their personal accounts using the required information

- vi. Username
- vii. Passwod

5.3. Users who have logged in can save an article or a news in the search results

5.4. Users who have logged in can view their profile with all the articles and news that they saved

5.5. Users who have logged in can subscribe an article to get notifications about new similar articles and news

5.6. Users who have logged in can get notifications about the latest articles and news which have the same keyword as the subscribe articles and news

Platform Use Cases:

Use Case 1	User views a list of reports in a certain period of time, about a specific disease(optional) and location(optional).
Related Requirements	1, 3
Actor	A guest user/logged in user
Use Case Overview	The user fills in the search box on home page to make a request and views all the reports in report section on result page.
Subject Area	Seach form, result page(reports section)
Trigger	The user are interested to see the disease reports about a disease

	or location in a certain period of time.
Precondition	User is on the homepage of website

Basic Flow: View the disease reports

Description	This scenario describes the situation where a user can view a list of reports after input information needed for search
1	The user enters the start date, end date, disease name (optional) and location (optional) into the search form
2	The user clicks on "Search"
3	The system redirects the user to the result page
4	The system gets the a list of articles from calling Cidrap API, EpiPro API based on the input from the user
5	The user clicks on the "Articles" section on result page
6	The system displays the list of articles under "Articles" section
7	The user clicks on "More Info" button to access more details about the report
Termination outcome	The disease report is displayed successfully

Post Conditions: A list of articles based on the user's requirement are displayed under the section of "Articles" on result page.

Use Case 2	User views a list of latest news in a certain period of time, about a specific disease.
Related Requirements	1.2, 1.4, 4
Actor	A guest user/logged in user
Use Case Overview	The user fills in the search form on home page to make a request and views all the latest news in the news section on result page.
Subject Area	Search form, result page(news section)

Trigger	The user are interested to see the latest news about a disease in a certain period of time.
Precondition	User is on the homepage of website

Basic Flow

Description	This scenario describes the situation where a user can view a list of reports after input information needed for search
1	The user enters the start date, end date, disease name and location (optional) into the search form
2	The user clicks on "Search"
3	The system redirects the user to the result page
4	The system gets the a list of news from calling Google News API based on the input from the user
5	The user clicks on the "News" section on result page
6	The system displays the list of articles under "News" section
7	The user clicks on "More Info" button to view more details about the news
Termination outcome	The news is displayed successfully

Post Conditions: A list of news based on the user's requirement are displayed under the section of "news" on result page.

Use Case 3	User accesses a line chart in a certain period of time, about a specific disease.
Related Requirements	1.2, 1.4, 2.1
Actor	A guest user/logged in user
Use Case Overview	The user fills in the search form on home page to make a request and accesses the line chart in graph section on result page.
Subject Area	Seach form, result page(graph section)
Trigger	The user are interested to see the trend of the development of a specific disease.
Precondition	User is on the homepage of website

Basic Flow

Description	This scenario describes the situation where a user can access a line chart after giving information needed for generating the line chart
1	The user enters the start date, end date, disease name, location name(optional) into the search form
2	The user clicks on “Search”
3	The system redirects the user to the result page
4	The system gets the a list of reports from calling Cidrap API, EpiPro API based on the input from the user
5	The system generates a line chart based on the list of the reports
6	The user clicks on the “Graph” section on result page
7	The system displays the line chart under “Graph” section
Termination outcome	The line chart is displayed successfully

Post Conditions: A line chart showing the trend of the development of the disease is displayed under the section of “Graph” on result page.

Use Case 4	User accesses a donut chart in a certain period of time, about a specific location
Related Requirements	1.1, 1.3, 2.2
Actor	A guest user/logged in user
Use Case Overview	The user fills in the search form on home page to make a request and accesses the donut chart in graph section on result page.
Subject Area	Search form, result page(graph section)
Trigger	The user are interested to see the distribution of diseases in a specific location
Precondition	User is on the homepage of website

Basic Flow

Description	This scenario describes the situation where a user can view a donut graph after typing in information needed for generating the graph
-------------	---

1	The user enters the start date, end date and location into the search form
2	The user clicks on “Search”
3	The system redirects the user to the result page
4	The system gets the a list of articles from calling Cidrap API, EpiPro API based on the input from the user
5	The system generates a donut graph based on the list of the reports
6	The user clicks on the “Graph” section on result page
7	The system displays the donut graph under “Graph” section
Termination outcome	The donut graph is displayed successfully

Post Conditions: A donut graph showing the distribution of the disease about a specific location is displayed under the section of “Graph” on result page.

Use Case 5	Users can save a particular article that they are interested in
Related Requirements	1, 5.1, 5.2, 5.3
Actor	A logged in user
Use Case Overview	On report detail page, click on “Save Articles” button to save a particular article
Subject Area	“More Info” page
Trigger	The user wants to save a particular article to access it later
Precondition	After searching for articles, on result page, the user clicks on the article section and clicks on “More Info” button to see the details of a report

Basic Flow:

Description	This scenario describes the situation where a user wants to save a particular article after viewing the details of the report
1	The user clicks on “Save” button
2	The system saves the information of the article and the user to database

3	The system displays a message to the user “You have successfully saved the article”
Termination outcome	The article is saved successfully

Post Conditions: An article of interest of the user is saved for that user.

Use Case 6	Users can view a list of articles they have saved
Related Requirements	5.1, 5.2, 5.4
Actor	A logged in user
Use Case Overview	After log in, the user can view a list of saved articles and reopen those articles
Subject Area	User profile page, home page
Trigger	The user wants to access some articles being saved before
Precondition	The user has logged in

Basic Flow:

Description	This scenario describes the situation where a user wants to access a list of articles he/she has saved before
1	The user clicks on “profile” button on home page
2	The system redirect the user to the profile page
3	The system displays a list of saved articles on the profile page
4	The user clicks on “More Info” button to view the details of the report
Termination outcome	A list of saved articles are displayed

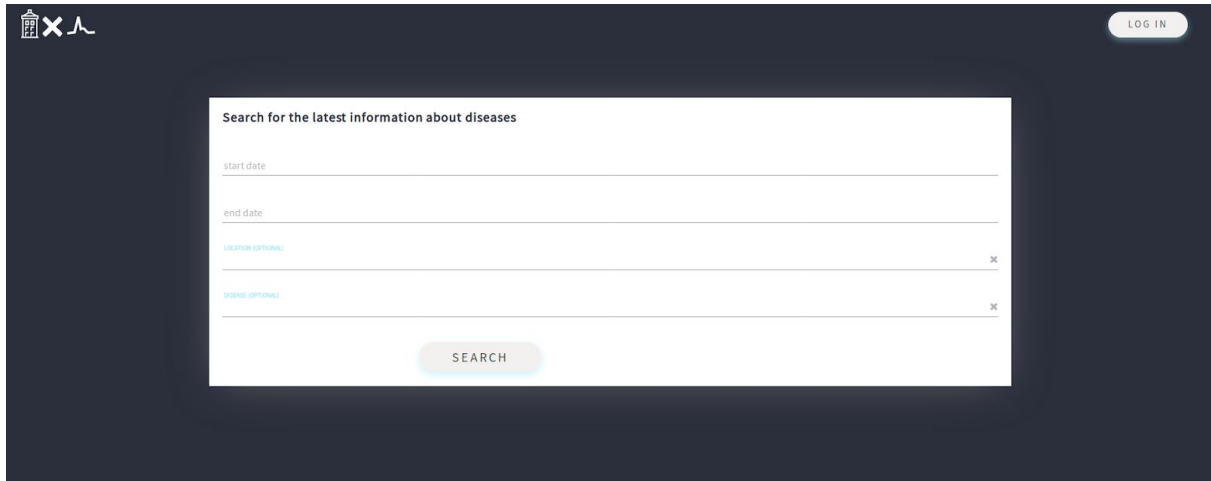
Post Conditions: On user profile page, the list of saved articles are displayed

Design

Our analytics platform is a single page application consisting of three main components: Search, Results, and Dashboard.

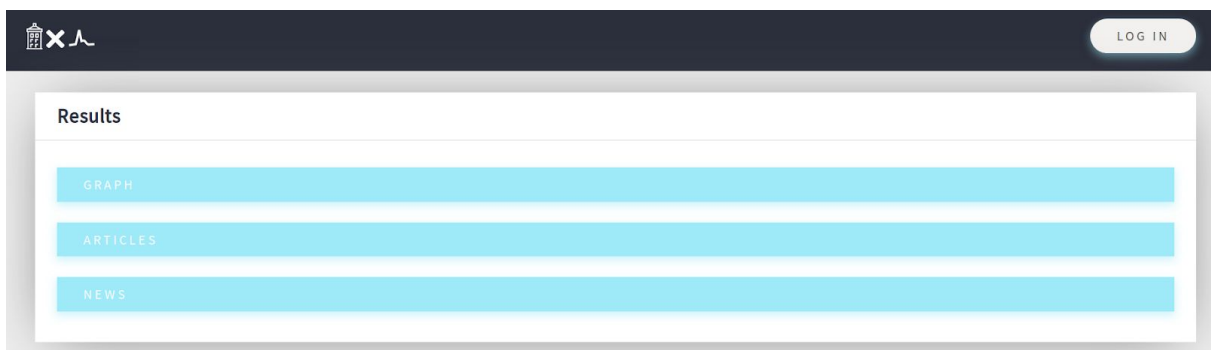
Search

This is the homepage of our webapp. Here, users are able to search for articles within a range of dates. They can also input additional optional information: key terms and location, to further refine their search.

A screenshot of a web application's search homepage. The background is dark blue. In the top left corner, there are three small white icons: a building, a cross, and a pulse line. In the top right corner, there is a light blue button with the text "LOG IN". In the center, there is a white rectangular search form. At the top of the form, it says "Search for the latest information about diseases". Below this, there are four input fields: "start date", "end date", "LOCATION (OPTIONAL)", and "DISEASE (OPTIONAL)". Each of the last three fields has a small "x" icon to its right. At the bottom of the form, there is a light blue button with the text "SEARCH".

Results

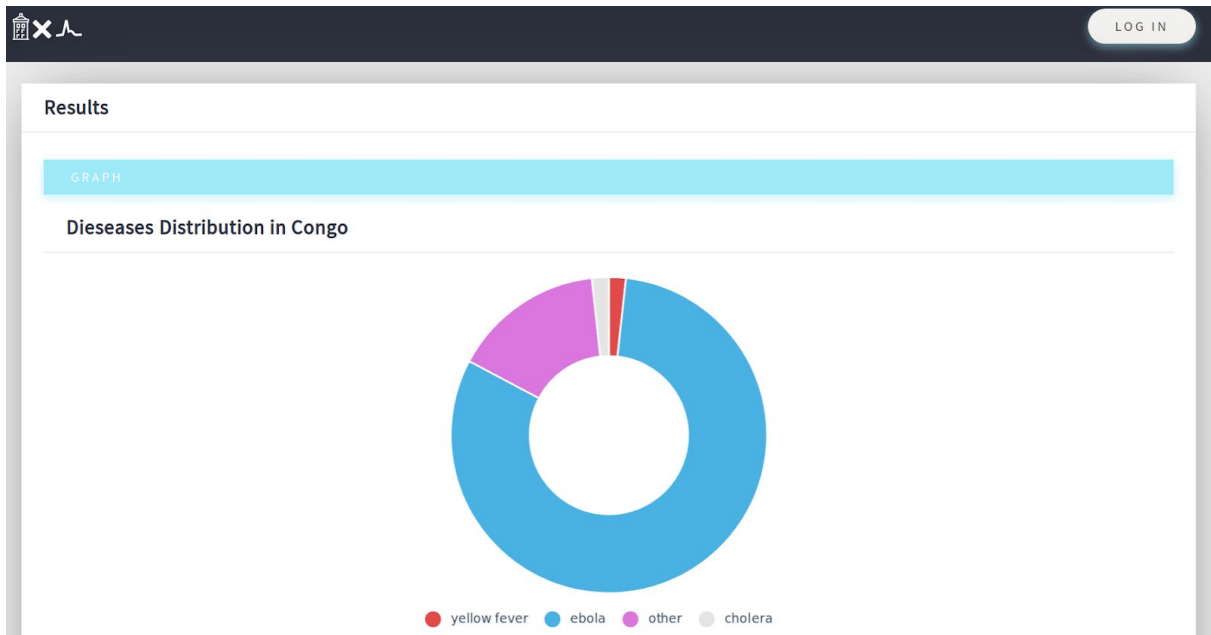
After a search is made, the results are displayed in three sections: Graph, Articles and News.

A screenshot of a web application's results page. The background is dark blue. In the top left corner, there are three small white icons: a building, a cross, and a pulse line. In the top right corner, there is a light blue button with the text "LOG IN". Below the header, there is a light gray rectangular area. Inside this area, at the top, is a white box with the text "Results". Below this, there are three horizontal light blue bars. The first bar has the text "GRAPH", the second bar has the text "ARTICLES", and the third bar has the text "NEWS".

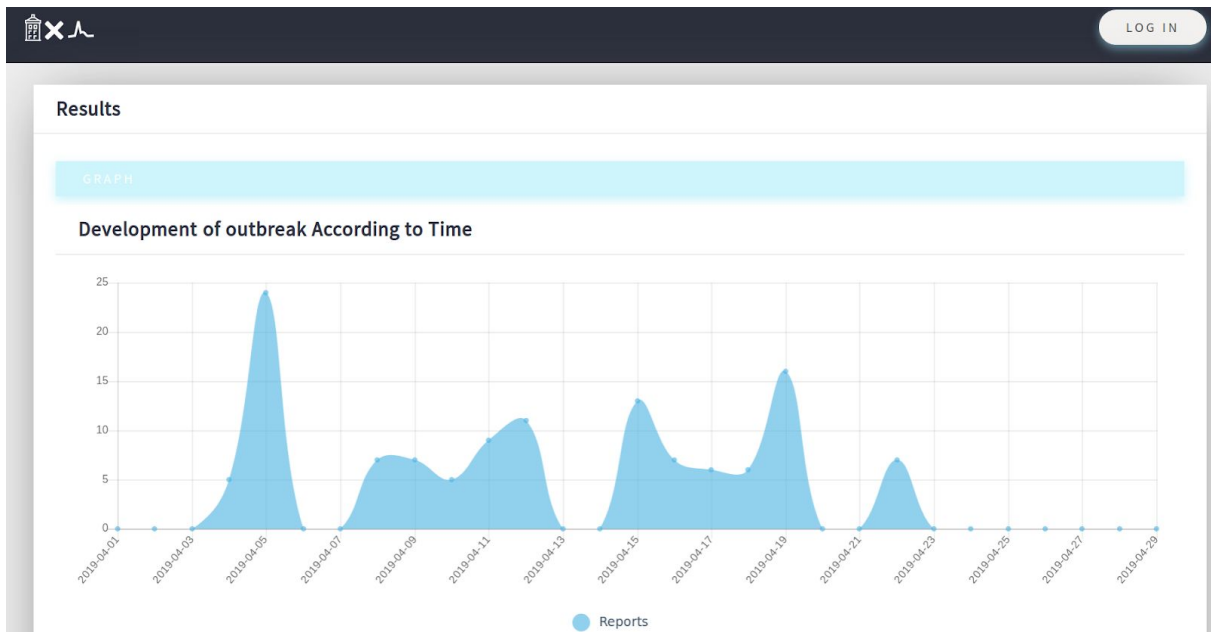
Graph

The Graph section contains a graph related to the search data.

If only the location is provided, and there are no key terms, a pie chart of the proportions of diseases in that location over the specified time period is displayed.

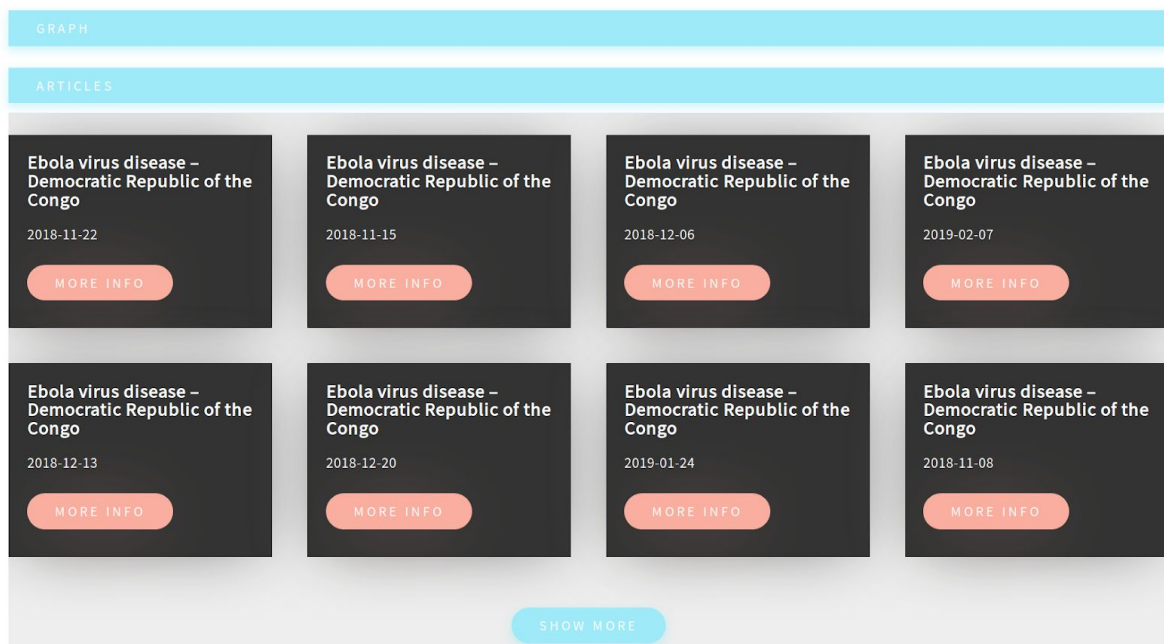


If key terms are included in the search, a line chart of the number of reports related to the key terms over time is displayed.

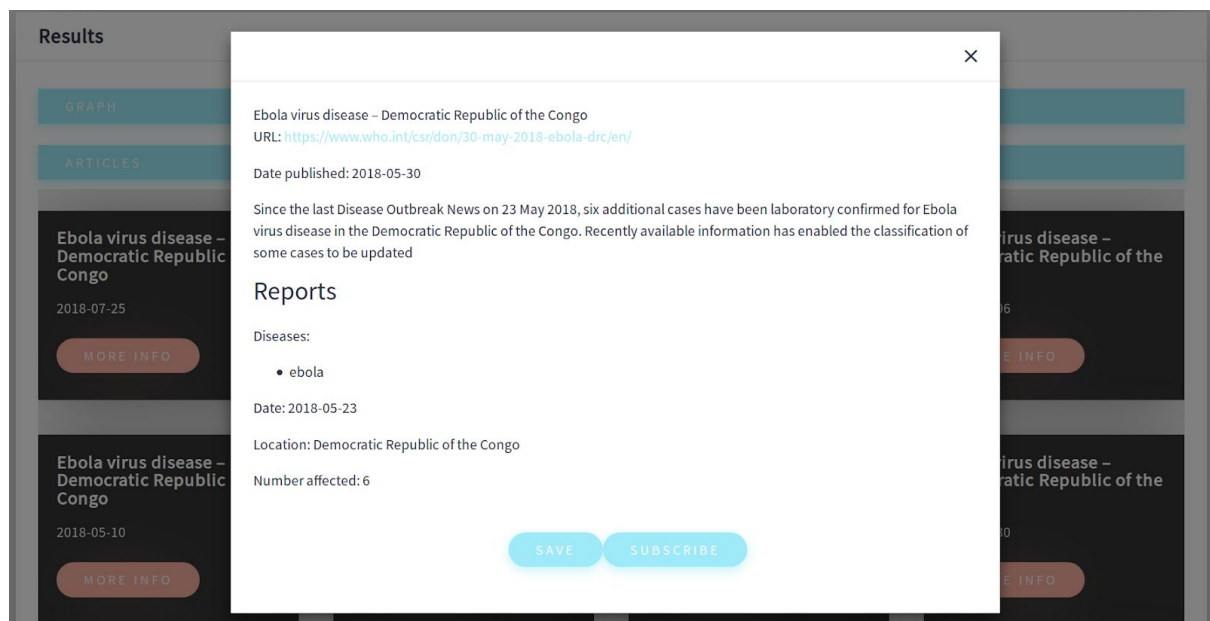


Articles

The first 8 articles that satisfy the search inputs are shown. Clicking on the “Show More” button will retrieve 8 more articles.



Clicking on the “More Info” button on an article will open up a modal containing a summary of that article.



If the user is logged in, they have the ability to save an article or subscribe.

News

The most recent 8 Google News articles relevant to the search inputs are displayed in the News section. Logged in users can also save and subscribe to news articles.

GRAPH

ARTICLES

NEWS

WHO Director-General visits DR Congo to address health workers' concerns
2019-04-28
[MORE INFO](#)

2017 Ebola outbreak caused by contact with bush meat - Healio
2019-04-28
[MORE INFO](#)

PRO/AH/EDR> Ebola update (41): Congo DR (NK, IT) cases, summary, WHO
2019-04-28
[MORE INFO](#)

Half of Europeans wrongly believe vaccines often cause serious side-effects
2019-04-28
[MORE INFO](#)

Nigerian health care critical
2019-04-28
[MORE INFO](#)

The Ebola outbreak in Congo is getting worse
2019-04-28
[MORE INFO](#)

Science News Briefs From All Over
2019-04-27
[MORE INFO](#)

Science News Briefs From All Over
2019-04-27
[MORE INFO](#)

Dashboard

A logged in user can access their dashboard by clicking on their profile picture and the “My Profile” button. The user's saved articles (including news articles) are displayed here.

SAVED ARTICLES

Ebola virus disease – Democratic Republic of the Congo
2018-05-30
[MORE INFO](#)

Ebola infects 17 more in DRC outbreak - CIDRAP
2019-04-17
[MORE INFO](#)

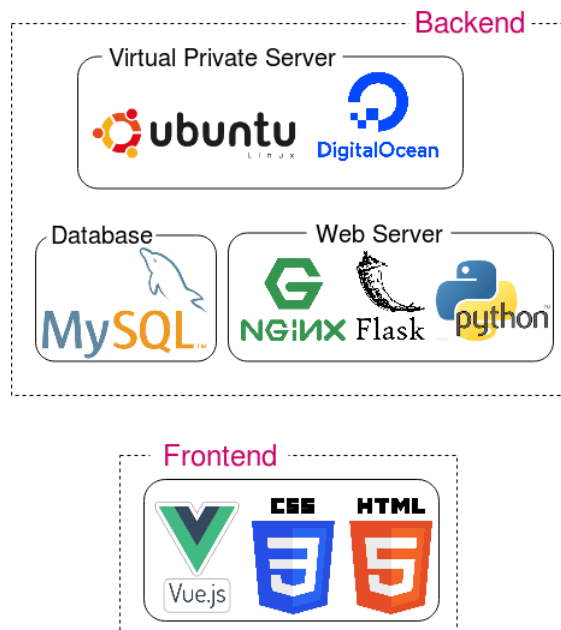
PRO/AH/EDR> Ebola update (38): Congo DR (NK, IT) cases, summary, WHO, response, research
2019-04-17
[MORE INFO](#)

Human infection with avian influenza A(H7N9) virus – China: Update
2018-09-05
[MORE INFO](#)

NK cells negatively regulate CD8 T cells via natural cytotoxicity receptor (NCR) 1 during LCMV infection
2019-04-17
[MORE INFO](#)

Software Architecture (Application)

APPLICATION



Programming Language

For our application, we used Python in conjunction with the Flask framework for backend. Since we had used Python and Flask for our API, we decided to continue applying this for our application to save time on setting up the server, as well as for compatibility with the stacks we had for the project. We used Vue.js for frontend, so the backend would only need to handle user login and registration to contact database, and saved articles for users in the database.

For frontend, we decided to use Vue.js for friendly user experience and interface.

Development environment

The next decision was whether to use Django or Flask framework. Django has a lot of built in features, making it easy to start developing. Flask is more lightweight, but allows for greater flexibility. Also, Flask has many useful libraries for building the API. For example, flask has a library called flask_restful that's designed for building RESTful api. This is why we chose Flask.

Python libraries

Since we are using flask framework, we are going to use those libraries to build the api: flask, flask_restful, flask_jwt, security, user

To make the scraper, we are going to use those libraries:

BeautifulSoup, requests, json

We also use other libraries to make new independent thread to update database daily such as:

threading, time, datetime

Database

We decide to use MySQL as our database for the application, to store user accounts, and user's saved articles and subscription details for notifications.

Deployment

After API implementation in backend, we decided to deploy the application on DigitalOcean Virtual Private Server running Linux system (Ubuntu 18.04) with all built-in dependencies required to run our application (Python, MySQL database, etc.). We can also register a domain name and have it point to the address of our VPS on DigitalOcean.

Reasons for choosing DigitalOcean VPS are:

- We get educational pack from Github for UNSW students credit
- Some of our members had experience running applications from DigitalOcean VPS
- DigitalOcean VPS provides many features, in which it can be run with dependencies and the environment of our choice, so we can run the application in the same environment that we implemented it in locally

API Integration

We used our API to search for articles based on the user's inputs on the search page. The four parameters: start date, end date, key terms and location were extracted from the Search form, and converted to the format required by the API.

Then the axios library was used to perform a GET request to the '/articles' endpoint, with these parameters passed in as a dictionary. The API response was a json, containing a list of articles.

This list of articles was saved as a field "cidrap_res" in the Vue component for results. Vue made it simple to iterate through this list, access fields of the json objects, and generate HTML to create a different card for each article shown in the search results.

Generating Graphs

In our website, we used vuestic charts from Vue.js to generate two types of graphs the line charts and the donut charts according to different input fields. The line charts are made to display the trends of the development of a specific disease according to time, while the donut charts are for the distribution of different diseases in a specific location or all over the world.

When the search fields only include the time period, we first get a list of all the diseases from the search results. Then we compute the number of reports for each disease in that time period regardless of the location. We store the report numbers in the order of how the diseases are stored. When generating the donut charts, the list of diseases is used as the labels and the list of report numbers is used as the data.

Similarly, when the search fields include the time period and the location, we just compute the number of reports for each disease in the inputted location and time period. Then generate the disease distribution in the specified location using the donut chart.

On the other hand, when the search fields contain a key term, we generate a line chart to show the trends of the development of the specified disease. We first get a list of all the dates from the start date to the end date from the search results. Then we compute the number of reports for the inputted disease on each day. By storing the report numbers in the orders of the dates, we can then set the labels of the chart as the dates and the data as the list of report numbers.

When the user inputs the time period and both the key term and the location, we simply count the reports for the specified disease in the specified location on each day and generate a line chart from the data as well.

Other APIs used

EpiPro (WHO)

Our API extracted data from CIDRAP, but we wanted to include another data source for disease articles to expand the scope of our data. We decided on WHO because it is a data source that is well regarded for accurate disease information. EpiPro's API seemed to have good information, so we used it.

The process of calling their API was quite similar to using our own API. We made a GET request to `"/reports/filter"`, using the same parameters from the user's input on the search page.

The y change that was needed was that this API had slightly different

- Mostly similar to our API
- APIs had slightly different formats
 - `event number_affected` (ours) vs `number-affected`
 - `syndrome` as list (ours) vs as string

Google News

Google News API is a powerful news API that can provide live articles over 30,000 news resources and blogs and it also has a very beautiful documentation. That's why we chose Google News API to provide the related news to our users.

The process of calling Google News API is quite similar to using our own API.

The base url is "<https://newsapi.org/v2/everything>". The parameters are start date, end date and keyterms. For example, if I want to search for news about disease ebola from Mar 25th to April 23th, then the url will become

"<https://newsapi.org/v2/everything>?q=ebola&apiKey=10c0fde8d4104133a3fcec0374729ade&from=2019-03-25&to=2019-04-23"