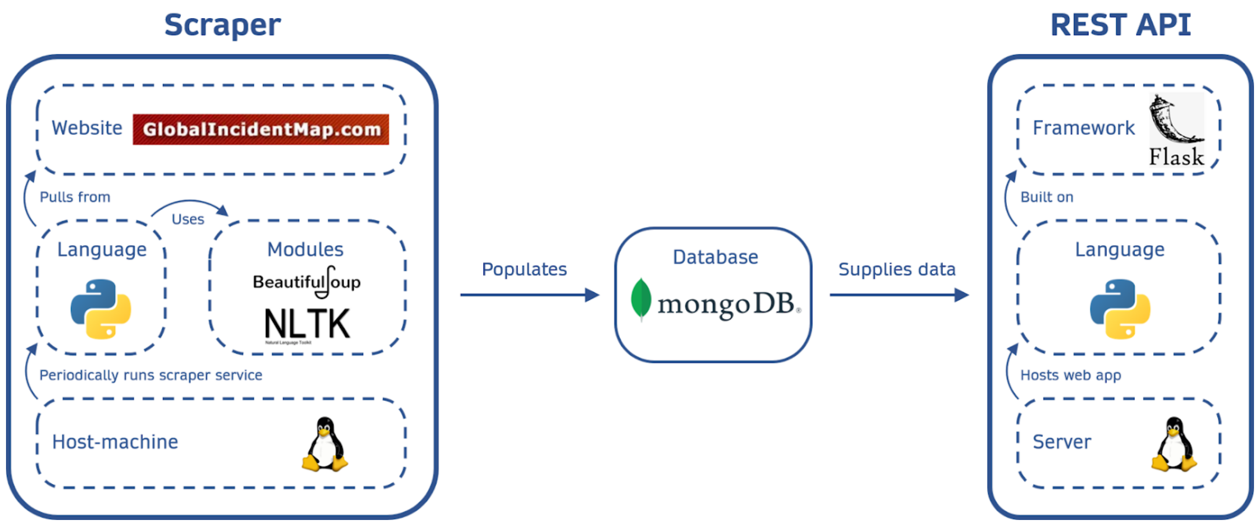
Design Details - Deliverable 1

1 Overall Design

1.1 Web Scraper

For the web scraper we will be using Python due to its ease of use. Being an interpreted language, developing a webscaper should be quick and simple. The primary module we will use is BeautifulSoup which allows us to parse and search html pages with ease. We will use other libraries and modules like regex, word2number and Natural Language Toolkit to extract more detailed information.

The site we are scraping (globalincidentmap) offers detailed reports sorted in chronological order. Consequently the scraper will be run daily parsing only data it has not seen before. It may be the case that there is more information to be found in the articles it links to. If this is the case then we will have the scraper explore those articles as well searching for keywords.



1.2 REST API

The API module will be developed using Python with Flask, a web app development framework. Using this framework a simple web application can easily be created that allows users to submit data to an endpoint of the website using a HTTP request. Each request will include a parameter that specifies what data the response should include. The web application will use the parameter to filter the records in the database and return the appropriate results.

The web scraper explained in 1.1, will populate a MongoDB database that the API will access for the data it will send as a response. MongoDB will be used since it stores data in a JSON format, the same format used throughout the application, making it easier to interact with. This web application will need to be hosted on a server, such as on a CSE machine, to provide the ability to run in web service mode.

1.3 Swagger

To aid with the use of our API by other teams we have decided to use Swagger. Swagger allows for clear documentation of our API, in an easy to read form. It also allows for someone to create dummy calls to our API through the Try It Out feature. Thus, another person is able to see how the calls work and what form they should be in. This aids with understanding as they are easily able to see what the calls should look like, thus easy integrate our API into their web app.

1.4 Web App

1.4.1 Frontend

The web app is where we can showcase our API data, so it is important to choose a frontend tech stack that enables us to quickly create a working app, while at the same time allowing for complex component design that can show off our features.

The framework we chose to use was React. This is for two reasons. Firstly, many of us have industry experience working with React, and so we can write the website at a faster pace. React also allows for higher complexity with its component system, enabling our team to go above and beyond, without being hindered by the framework.

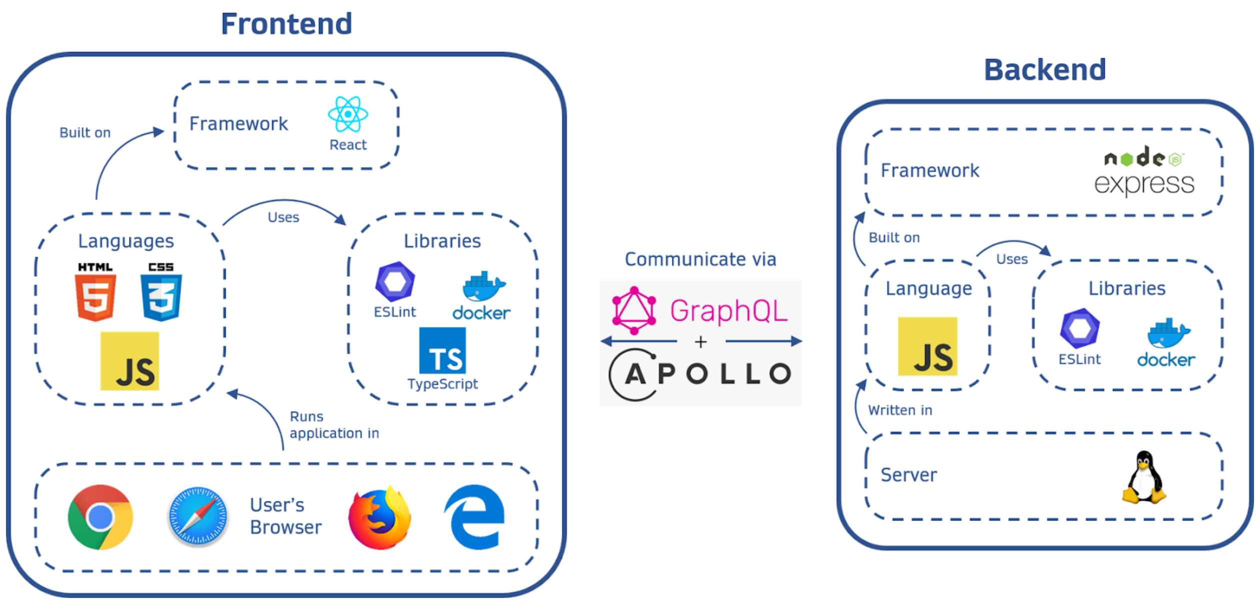
To communicate with the backend, we are using Apollo GraphQL. This is widely accepted to be the most straightforward way of communicating with the backend through a single endpoint, and allows our users to offload the processing of data to the database. This means that our website uses less power and loads at a faster rate than comparable websites using standard fetch requests. It also makes our endpoint much easier to use, as they only have to understand the schema for a single endpoint.

There are three supporting libraries that we are utilising: Typescript, ESLint and Docker.
Typescript allows us to typecast our javascript code, essentially reducing the time spent tracking bugs caused by javascript’s dynamic typing system.
ESLint allows us to standardise our code style, reducing the time it takes for new developers to understand what is going on. It also does this automatically, so no extra time is spent on formatting.

Docker allows us to standardise the testing and development of the frontend. This means that everyone who runs our code should have identical outputs, allowing for easier bug fixing. It also automates a lot of the setup, allowing more devs to quickly become active in the development workflow when needed.

1.4.2 Backend

Our backend will be quite lightweight, as most complex analysis and data will be retrieved from our public API. Due to this, we decided to use NodeJS Express, which is known for its flexibility and ease of use. Express also has a GraphQL library which we can use to serve our services from a single endpoint. Just like the frontend, we will use ESLint and Docker. Docker in particular will allow us to quickly run a test backend and frontend instance in conjunction, improving our testing capabilities.



2 API Design

2.1 Endpoint Design

Clients can interact with the API through a single endpoint via HTTP requests. These requests will use parameters to contain all the necessary information required by the API to calculate and send back the appropriate results. Depending on the data contained in the parameter, the API will filter the data stored in the database and return the requested data as a JSON object. The parameter must adhere to a strict structure made up of 4 pieces of information represented in the form: {“start_date”, “end_date”, “key_terms”, “location”}, where start_date and end_date are in the form “yyyy-MM-ddTHH:mm:ss”. If the parameter does not adhere to the required structure then the API will return an error result with an appropriate error code. Below is a table describing the possible response codes and the reason for them.

Response Code	**Reason**
200	Everything is ok and correct result is returned
400	There is an issue with the parameter
404	No data matching the parameter requirements could be found
405	The request method used is not allowed
500	The API had an issue processing the request

2.2 Example Interactions

Request Method	**Parameters**	**Response**	**Test**
GET	?start_date=2015-01-01 T00:00:00 &end_date=2018-01-23 T19:37:12 &key_terms=swineflu &location=punjab	{"date": "2015-02-15 17:33:00", "country": "Australia", "url": "http://www.abc.net.au/news/2015-02-16/pig -brucellosis/6116970", "coord": "-33.864, 151.205", "city": "New South Wales", "disease": "Brucellosis", "description": "[ABC.net.au]\u00a0 AUSTRALIA :: Call for pig dogs to be tested for brucellosis\n\r\n\"Veterinarians in the Hunter Valley and North West regions of New South Wales are calling on pig dog owners to be be on the lookout for signs of brucellosis after several dogs tested positive for the disease.\"\n\r\nRead Full Article At :: http://www.abc.net.au/news/2015-02-16/pig- brucellosis/6116970"}	Normal Request
GET	?end_date=2018-01-23 T19:37:12 &key_terms=swineflu &location=punjab	400 status code with no JSON object returned due to the parameter not meeting the required structure	Missing Start Date

GET	?start_date=2015-10-01 T08:45:10 &end_date=2016-11-01 T19:37:12 &key_terms=covid19 &location=sydney	404 status code with no JSON object returned due to the API not having any data that meets the parameter requirements	No Data
POST	?start_date=2015-01-01 T00:00:00 &end_date=2018-01-23 T19:37:12 &key_terms=swineflu &location=punjab	405 status code with no JSON object returned due to the request method not being GET	Post Request
GET	This could happen independent of the supplied parameters, as a 500 status code is a catch for uncaught errors in the server.	500 status code with no JSON object returned due to an API error, possibly due to a missed edge-case	N/A

3 Implementation Choices

3.1 Implementation Language

The main languages that we have decided to go with are Python and JavaScript.

3.1.1 Python

We are using Python for the web scraper and RESTful API due to the simplicity of the language. It allows us to focus on the implementation of the services, rather than getting bogged down in the complex structures that other languages, such as C or an Object Oriented language such as Java, poses. This also helps to increase the reliability of our code, which allows for seamless collaboration throughout the team.

Also, Python allows us to use a framework such as Flask, which handles a large amount of the overhead for creating a Web API. This allows us to yet again focus on the implementation of the API, increasing our productivity.

We understand that the choice to go with Python will reduce the speed of our program, as Python is inherently slower than a language like C.However, the benefits of increased productivity and readability outway the negative performance effects.

3.1.2 JavaScript

We have decided to use JavaScript, coupled with HTML and CSS, as our main frontend language as it has become an industry standard for web applications. Also, we are able to use frameworks such as React to help with the development of our frontend. React offers standard conventions and structuring that will help in the continued development of our web application. It will allow us to add new features and pages with ease, in a way that appears integrated with the rest of the site.

We will also be using NodeJS as our backend for the web application, coupled with Expressed, as this is a well used tech stack. The simplicity of Express will allow for the backend to be created simply and efficiently, and will also help in adding extra features as the project goes on.

3.2 Development and Deployment Environment

For our deployment environment we have decided to go with Linux. The reason we are using Linux is that it is lightweight and simple to set up for the small services that we are creating. Also, it is a common choice for services like ours so there is a large amount of documentation and help online, so we will be able to easily setup the services and diagnose issues when they arise.

However, all our frameworks work cross platform so during development we can all use them independent of our own environments.This speeds up development, as we do not have to individually configure our environments to help run this process. And with the introduction of Docker we will be able to perform the setup for our environments quickly.

We have decided to deploy our project onto Google Cloud Platform. The reason is that its highly versatile in what can run on the service, as it has both virtual machines for running the web scraper and app deployment for our API and Web App. It also supports linux deployment, which is our chosen environment.

3.3 Libraries

3.3.1 Typescript

Typescript is an open source library that adds typing to traditional javascript. This improves clarity, as a developer can discern more information from any given variable, and also reduces room for error when variables are misinterpreted. The overarching reason we added typescript was to improve our quality, and pace, of code written.

3.3.2 Apollo

Apollo is the industry-standard GraphQL implementation for services written in javascript. Instead of using a traditional REST service, we can access the backend through a single endpoint, allowing our users to offload most computation to our services. This means that our app uses less power and loads at a faster rate than comparable websites using standard fetch requests. It also makes our endpoint much easier to use, as they only have to understand the schema for a single endpoint.

3.3.3 Beautiful Soup

Beautiful Soup is an open source library which allows aids in the extraction of data from HTML files. It creates a parse tree of the file which can be easily navigated, searched and modified in idiomatic ways. This is incredibly useful for scraping data off websites as it grants the ability to use extract data based on its HTML structure.

3.3.4 Natural Language Toolkit (NLTK)

NLTK is a collection or libraries and programs which help programs reason about natural language texts. These include tagging, machine learning, parsing, probability, tokenization, stemming and chunking of texts. NLTK will be used to find more detailed information from the articles and descriptions.

3.3.5 Requests

An open source library that makes sending HTTP requests very easy, abstracting most of the required functionality required with useful methods. Using this library's 'get' method along with the API's URL and a JSON object parameter, a HTTP GET request will be sent to the API and a data object returned that can be converted into a JSON object using the library's 'json' method, all in one convenient line.

3.3.6 Flask-PyMongo

Flask-PyMongo is an open source library that bridges Flask and PyMongo and provides some convenience helpers. PyMongo provides the connection to the MongoDB database used by the API.

Design Details - Deliverable 2

1 Final Architecture

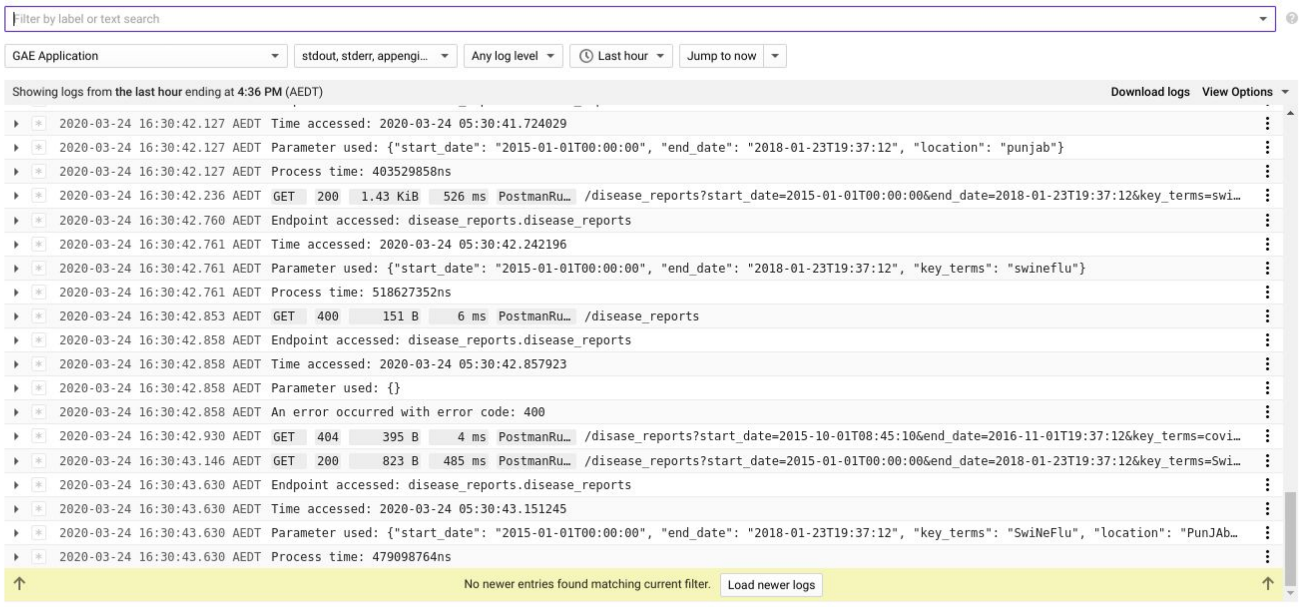
The final architecture that we developed is very similar to the architecture that we planned out in deliverable 1. However, there were some changes during implementation, such as additional features to improve the user experience and make it easier for us to monitor the API. The details of each component's architecture are described below.

We used MongoDB as our database, forming the bridge between the web scraper and the API. This allowed for data to be kept in a useful format, json. The web scraper could format the data it gets from the website into a JSON object and easily pass it into the database. From there the API could retrieve the data and perform little processing on it before sending it in the HTTP response.

The web scraper’s final architecture is very similar to what was described in deliverable 1. It uses beautiful soup to parse HTML and regex for keyword searches and string replacements. Finally pymongo is used to push suitable data to MongoDB from which the api will query.

The REST API uses Python and Flask to create our functional endpoint allowing users to make GET requests for our data. An additional module used was graphene, which allowed us to implement an optional graphQL endpoint for any users who preferred non-REST api’s. The format of queries and examples are documented in Swagger. Another addition was an endpoint that allows us to visually monitor all API requests and their performance. This was a huge improvement as it provides a graphical way to understand the success of our API rather than using the large amounts of text provided by log files.

Finally, all of the above services are hosted using the Google Cloud Platform, a single solution to meet all our web hosting needs. It provided capability to both host the API as an application, and a virtual machine through which we were able to run the web scraper remotely. This platform also provided a convenient location for the API logs, enabling us to search through the logs, finding each user’s request to the API and any errors that occurred. An example of the logs can be found in the image below.



Filter by label or text search
GAE Application
stdout, stderr, appengi...
Any log level
Last hour
Jump to now
Showing logs from the last hour ending at 4:36 PM (AEDT)
Download logs
View Options

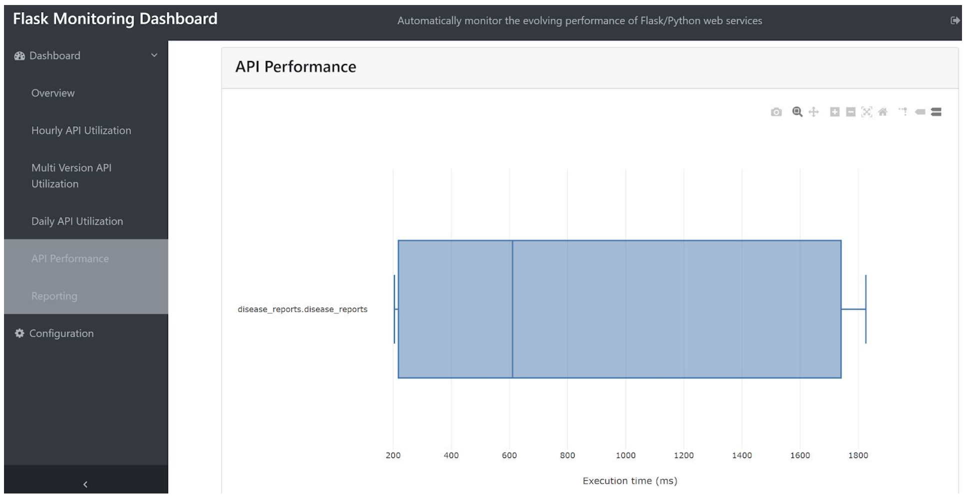
```
2020-03-24 16:30:42.127 AEDT Time accessed: 2020-03-24 05:30:41.724029
2020-03-24 16:30:42.127 AEDT Parameter used: {"start_date": "2015-01-01T00:00:00", "end_date": "2018-01-23T19:37:12", "location": "punjab"}
2020-03-24 16:30:42.127 AEDT Process time: 403529858ns
2020-03-24 16:30:42.236 AEDT GET  200  1.43 KiB  526 ms PostmanRu…  /disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:12&key_terms=swi…
2020-03-24 16:30:42.760 AEDT Endpoint accessed: disease_reports.disease_reports
2020-03-24 16:30:42.761 AEDT Time accessed: 2020-03-24 05:30:42.242196
2020-03-24 16:30:42.761 AEDT Parameter used: {"start_date": "2015-01-01T00:00:00", "end_date": "2018-01-23T19:37:12", "key_terms": "swineflu"}
2020-03-24 16:30:42.761 AEDT Process time: 518627352ns
2020-03-24 16:30:42.853 AEDT GET  400     151 B    6 ms PostmanRu…  /disease_reports
2020-03-24 16:30:42.858 AEDT Endpoint accessed: disease_reports.disease_reports
2020-03-24 16:30:42.858 AEDT Time accessed: 2020-03-24 05:30:42.857923
2020-03-24 16:30:42.858 AEDT Parameter used: {}
2020-03-24 16:30:42.858 AEDT An error occurred with error code: 400
2020-03-24 16:30:42.930 AEDT GET  404     395 B    4 ms PostmanRu…  /disase_reports?start_date=2015-10-01T08:45:10&end_date=2016-11-01T19:37:12&key_terms=covi…
2020-03-24 16:30:43.146 AEDT GET  200     823 B  485 ms PostmanRu…  /disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:12&key_terms=Swi…
2020-03-24 16:30:43.630 AEDT Endpoint accessed: disease_reports.disease_reports
2020-03-24 16:30:43.630 AEDT Time accessed: 2020-03-24 05:30:43.151245
2020-03-24 16:30:43.630 AEDT Parameter used: {"start_date": "2015-01-01T00:00:00", "end_date": "2018-01-23T19:37:12", "key_terms": "SwiNeFlu", "location": "PunJAb…
2020-03-24 16:30:43.630 AEDT Process time: 479098764ns
```

Image of Google Cloud Platforms logs

2 Choice of Implementation

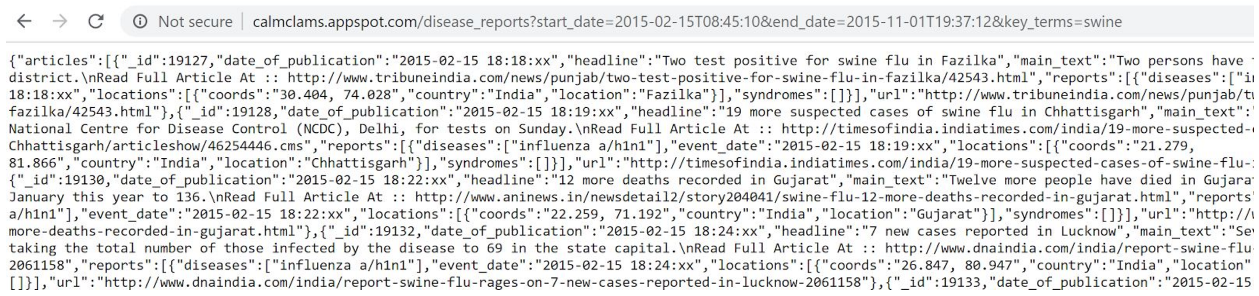
2.1 REST API

As planned, the final API was developed using Python with the web app development framework Flask. This allowed for a quick development and deployment as the design was kept simple and small scale with one endpoint for functionality. What did change however was the addition of a new endpoint used to monitor resource utilisation visually, as shown in the image below. This won't affect how users interact with our API but will allow us to understand how they do and how well our API is performing.



Screenshot of Resource Utilisation Dashboard

The functional endpoint still involves users submitting parameters however they are no longer in the form of a JSON object and are instead passed as url query arguments. This made it easier to test as the API could be queried directly in the browser rather than using some testing service, an example can be seen in the image below. It also meant that our API adhered to industry standards since we are using the GET HTTP request and no longer sending a data object which would only be appropriate with a POST HTTP request.



Not secure | calmclams.appspot.com/disease_reports?start_date=2015-02-15T08:45:10&end_date=2015-11-01T19:37:12&key_terms=swine

```
{"articles":[{"_id":19127,"date_of_publication":"2015-02-15 18:18:xx","headline":"Two test positive for swine flu in Fazilka","main_text":"Two persons have
district.\nRead Full Article At :: http://www.tribuneindia.com/news/punjab/two-test-positive-for-swine-flu-in-fazilka/42543.html","reports":[{"diseases":["i
18:18:xx","locations":[{"coords":"30.404, 74.028","country":"India","location":"Fazilka"}],"syndromes":[]}],"url":"http://www.tribuneindia.com/news/punjab/t
fazilka/42543.html"},{"_id":19128,"date_of_publication":"2015-02-15 18:19:xx","headline":"19 more suspected cases of swine flu in Chhattisgarh","main_text":
National Centre for Disease Control (NCDC), Delhi, for tests on Sunday.\nRead Full Article At :: http://timesofindia.indiatimes.com/india/19-more-suspected-
Chhattisgarh/articleshow/46254446.cms","reports":[{"diseases":["influenza a/h1n1"],"event_date":"2015-02-15 18:19:xx","locations":[{"coords":"21.279,
81.866","country":"India","location":"Chhattisgarh"}],"syndromes":[]}],"url":"http://timesofindia.indiatimes.com/india/19-more-suspected-cases-of-swine-flu-
{"_id":19130,"date_of_publication":"2015-02-15 18:22:xx","headline":"12 more deaths recorded in Gujarat","main_text":"Twelve more people have died in Gujara
January this year to 136.\nRead Full Article At :: http://www.aninews.in/newsdetail2/story204041/swine-flu-12-more-deaths-recorded-in-gujarat.html","reports
a/h1n1"],"event_date":"2015-02-15 18:22:xx","locations":[{"coords":"22.259, 71.192","country":"India","location":"Gujarat"}],"syndromes":[]}],"url":"http://
more-deaths-recorded-in-gujarat.html"},{"_id":19132,"date_of_publication":"2015-02-15 18:24:xx","headline":"7 new cases reported in Lucknow","main_text":"Se
taking the total number of those infected by the disease to 69 in the state capital.\nRead Full Article At :: http://www.dnaindia.com/india/report-swine-flu
2061158","reports":[{"diseases":["influenza a/h1n1"],"event_date":"2015-02-15 18:24:xx","locations":[{"coords":"26.847, 80.947","country":"India","location"
[]}],"url":"http://www.dnaindia.com/india/report-swine-flu-rages-on-7-new-cases-reported-in-lucknow-2061158"},{"_id":19133,"date_of_publication":"2015-02-15
```

Example of Testing API Directly in the Browser

Finally, the API is now hosted using the Google Cloud Platform, providing the ability to run remotely and be accessible any time of the day. This service also provided a convenient way to view logs which was also added during implementation. They allow us to see every request made to the API, the parameters used and the response returned by the API, for better analysis and monitoring.

2.2 Web Scraper

The web scraper implementation, described in 2.1, was also the same as planned, populating a MongoDB database that the API has access to. MongoDB proved to provide an easy way to store and retrieve data as it follows the same JSON format as the parameters used to interact with the API and the objects returned from the API.

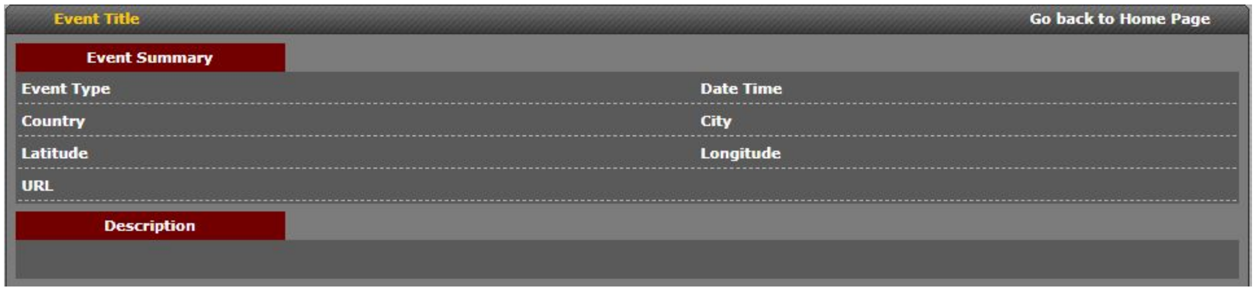
2.3 Testing

We chose to go with Postman for our tests. How we came to this choice is described below in section 3.4. We chose Postman because of the simpler, more comprehensive test suite we were able to make from it. For people who don’t know how to use postman, they can run a simple command on the collection to have the tests run, and those with basic knowledge of postman are able to use our tests to understand at a deeper level how our API works.

3 Challenges and Shortcomings

3.1 Web Scraper

The site (globalincidentmap) we are parsing has tens of thousands of records which can be accessed incrementally by changing the url. Unfortunately some of these records are empty and consequently have to be ignored. This is trivial however it makes knowing when the scraper has reached the latest record difficult. To get around this, the scraper runs hourly picking up on the last successfully scraped record and stops after seeing 30 consecutive empty records or after scraping 500 records. This allows the scraper to be constantly up to date while being able move past many consecutive empty records.



Event Title | Go back to Home Page

Event Summary

Event Type	Date Time
Country	City
Latitude	Longitude
URL	

Description

Image of a blank/empty record

The main issue we faced writing the web scraper was converting the event type field of the aforementioned records into a disease or syndrome as described in the spec. Ebola and Murburg were both part of the same event type and consequently had to be seperated. Also all types of influenza were combined into a single category. These issues were for the most part surmountable by searching for specific keywords. Other event types had to be completely ignored such as “Ricin” and “Suspicious or Threatening Powder” due to their being irrelevant or off topic (There were many diseases which primarily exist in animals).

3.2 REST API

One of the most challenging aspects of implementing the API was figuring out how to query the database. This was particularly difficult due to the significant lack of documentation for PyMongo, the python library used to interact with our MongoDB database. For example, it was problematic finding how to query the database for records that are within a particular date range. To solve this issue, we had to make educated guesses based on the limited documentation and adjust based on the results until we eventually made it work.

Another problem we had to overcome was manipulating the parameters into a form that was suitable for querying the database. Thankfully, Python provides a wide range of libraries and internal functionality for manipulating strings, so with a little research the problem of string manipulation was overcome. However, there remained the issue of understanding what format the data needed to be manipulated into, so that it matched the format of the database. This required communication with the team mate developing the web scraper as they made decisions of what the database format was.

As we had not had prior experience developing our own REST API's, the domain of possible cases was unknown to us. Through a process of trial and error we began to uncover the possible edge cases that could be presented. With use of our tests we were able to find and solve these edge cases effectively.

3.3 Swagger

The Swagger documentation of our API needed to clearly convey to users the structure and purpose of both the inputs and outputs of our system. Swagger is well suited to the latter, providing a schema section in which we can comprehensively define the structure of variables used, with examples, regex formats for strings, and lists of properties for objects. These schemas can then be referenced elsewhere, and Swagger automatically generates expandable examples for outputs with all the detail of the schema included.

However, when schemas are referenced from inputs, Swagger runs into two issues. The first is that the examples pulled from the schemas don't provide an immediately meaningful set of placeholders to the "Try it out" feature, which allows Swagger to make calls to our APIs, so we defined specific examples for each of the input values.

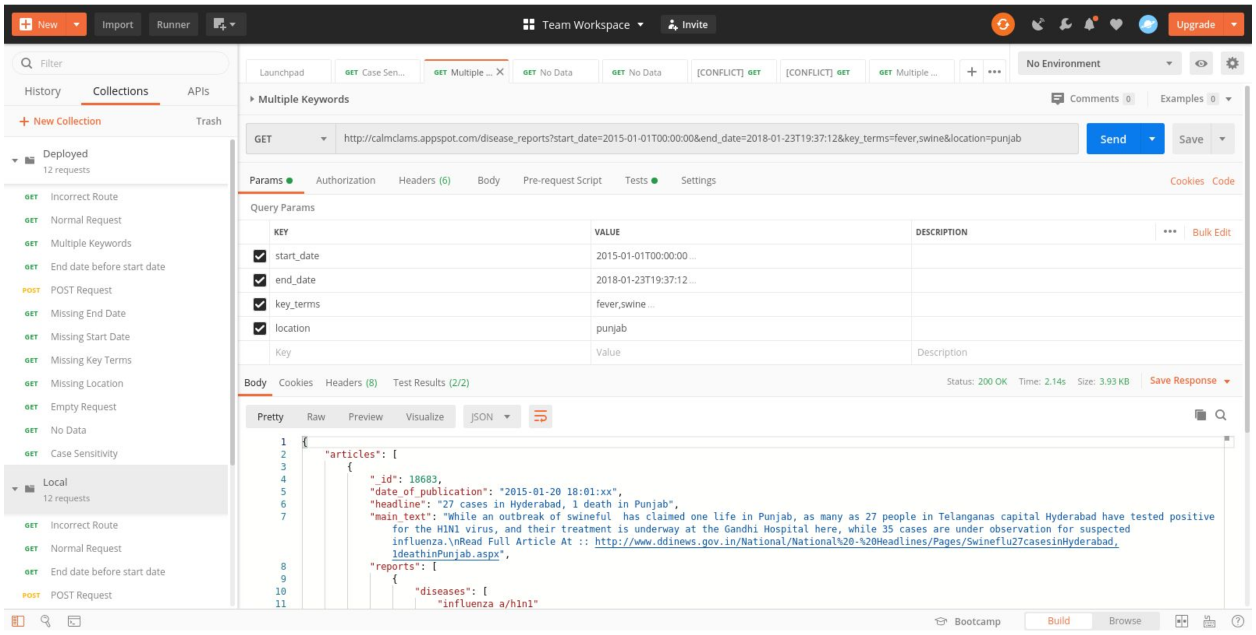
The second issue was that the schema details don't get shown with the inputs, making it difficult to tell what the structure of the inputs should be without reading through all the schemas to determine which applies. Anticipating that Swagger had some way of allowing us to reference this information, we went looking through the documentation for the Swagger application itself, and eventually found that the feature had been only recently programmed (just 2 weeks before the deliverable's deadline) and was scheduled to be included only in the next release. To ensure that users were still able to understand what each input should look like, we instead added descriptions and examples for the forms of each of the inputs to their descriptions, and where applicable directed users to the schema for the input's type for more information.

3.4 Testing

The original plan with the testing suite was to write a simple python script that would grab requests from text files and compare the response with an expected response. However, after complications with writing this small script, we realised that the response was more nuanced than hoped. A different order in the json object could cause the test to fail when it should pass.

The solution we came to here was to move all our tests to Postman. Postman has already handled all of these complications, and allows us to create more comprehensive tests much easier. Also, we are able to run postman tests on both the command line and the postman client, giving other teams a choice with how they want to interpret our tests.

See the testing document (https://github.com/z5122506/SENG3011-calmClams/blob/master/Reports/Testing%20Documentation.pdf) for testing details.



Deployed
12 requests

- GET Incorrect Route
- GET Normal Request
- GET Multiple Keywords
- GET End date before start date
- POST POST Request
- GET Missing End Date
- GET Missing Start Date
- GET Missing Key Terms
- GET Missing Location
- GET Empty Request
- GET No Data
- GET Case Sensitivity

Local
12 requests

- GET Incorrect Route
- GET Normal Request
- GET End date before start date
- POST POST Request

Multiple Keywords

GET http://calmclams.appspot.com/disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:12&key_terms=fever,swine&location=punjab

Query Params

	KEY	VALUE	DESCRIPTION
✓	start_date	2015-01-01T00:00:00 ...	
✓	end_date	2018-01-23T19:37:12 ...	
✓	key_terms	fever,swine ...	
✓	location	punjab	
	Key	Value	Description

Body Cookies Headers (8) Test Results (2/2)

Status: 200 OK Time: 2.14s Size: 3.93 KB

```
{
    "articles": [
        {
            "_id": 18683,
            "date_of_publication": "2015-01-20 18:01:xx",
            "headline": "27 cases in Hyderabad, 1 death in Punjab",
            "main_text": "While an outbreak of swineful  has claimed one life in Punjab, as many as 27 people in Telanganas capital Hyderabad have tested positive
                for the H1N1 virus, and their treatment is underway at the Gandhi Hospital here, while 35 cases are under observation for suspected
                influenza.\nRead Full Article At :: http://www.ddinews.gov.in/National/National%20-%20Headlines/Pages/Swineflu27casesinHyderabad,
                1deathinPunjab.aspx",
            "reports": [
                {
                    "diseases": [
                        "influenza a/h1n1"
```

Image of our test suite in Postman

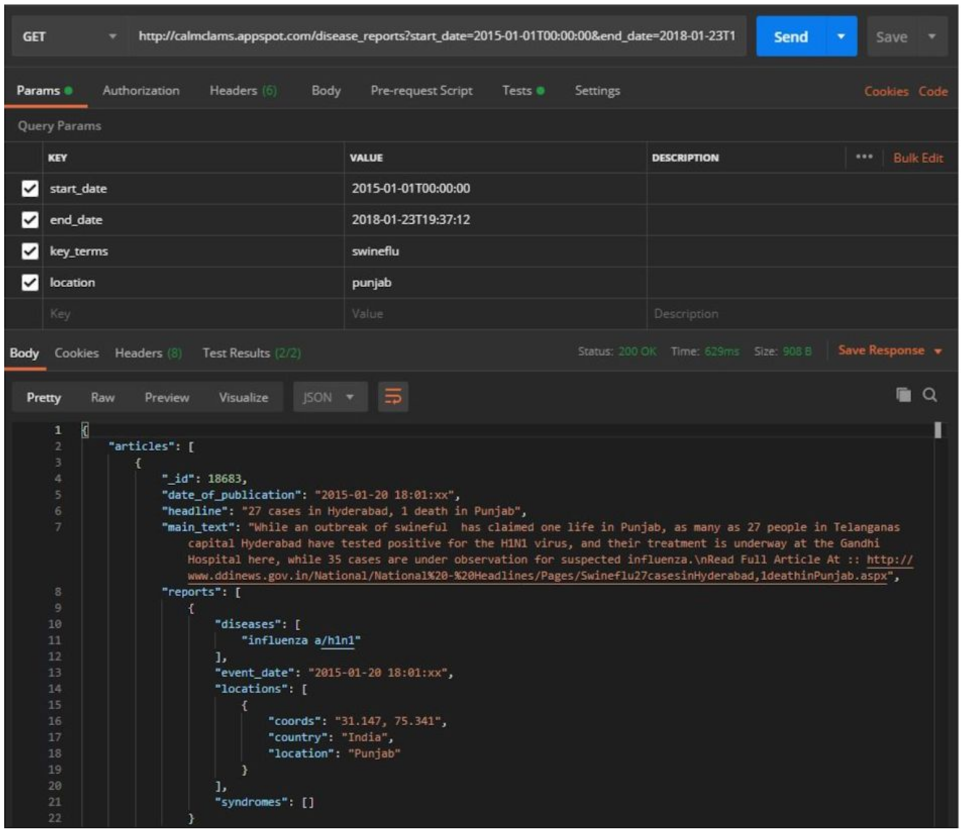
Design Details - Deliverable 4

1 Use Cases

1.1 API

Our API provides users a convenient way of retrieving content from the website; globalincidentmap, a site dedicated to “Displaying outbreaks, cases and deaths from viral and bacterial diseases”. We designed it to meet the requirements of our users and maintain simplicity while being flexible. Users request data from our API by adding relevant parameters to a query, allowing them to control what data they are returned. Apart from formatting specifics, the only required fields our API query has is a start and end date to specify the time range the reports returned must be in.

The following are screenshots from our Postman testing environment, demonstrating the different ways to use our API:



GET http://calmclams.appspot.com/disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T1

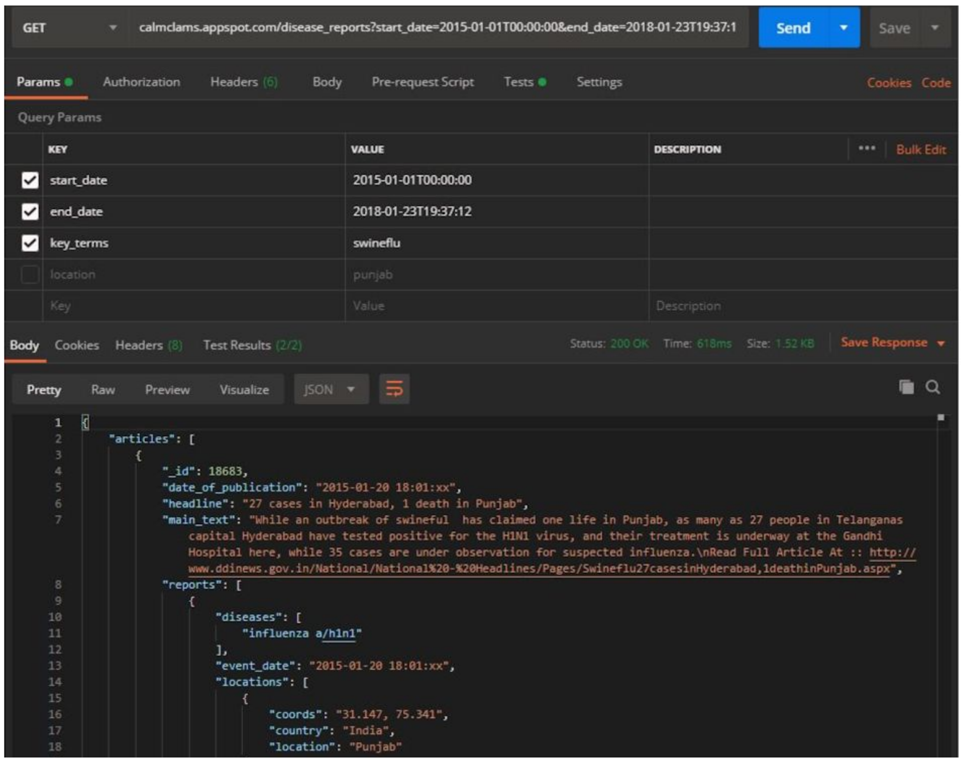
Query Params

	KEY	VALUE	DESCRIPTION
✓	start_date	2015-01-01T00:00:00	
✓	end_date	2018-01-23T19:37:12	
✓	key_terms	swineflu	
✓	location	punjab	

Status: 200 OK Time: 629ms Size: 908 B

```
{
    "articles": [
        {
            "_id": 18683,
            "date_of_publication": "2015-01-20 18:01:xx",
            "headline": "27 cases in Hyderabad, 1 death in Punjab",
            "main_text": "While an outbreak of swineful  has claimed one life in Punjab, as many as 27 people in Telanganas
              capital Hyderabad have tested positive for the H1N1 virus, and their treatment is underway at the Gandhi
              Hospital here, while 35 cases are under observation for suspected influenza.\nRead Full Article At :: http://
              www.ddinews.gov.in/National/National%20-%20Headlines/Pages/Swineflu27casesinHyderabad,1deathinPunjab.aspx",
            "reports": [
                {
                    "diseases": [
                        "influenza a/h1n1"
                    ],
                    "event_date": "2015-01-20 18:01:xx",
                    "locations": [
                        {
                            "coords": "31.147, 75.341",
                            "country": "India",
                            "location": "Punjab"
                        }
                    ],
                    "syndromes": []
                }
```

Above: Postman screenshot for the case where all parameters have values



```
GET calmclams.appspot.com/disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:1
```

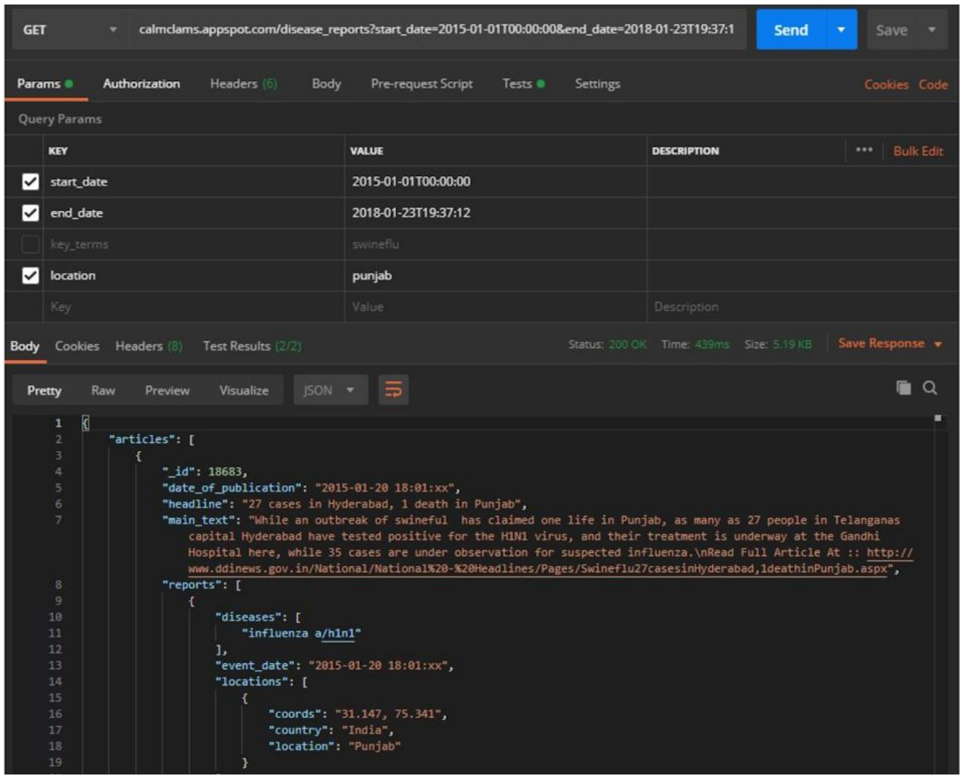
Query Params

KEY	VALUE	DESCRIPTION
✓ start_date	2015-01-01T00:00:00	
✓ end_date	2018-01-23T19:37:12	
✓ key_terms	swineflu	
location	punjab	

Status: 200 OK Time: 618ms Size: 1.52 KB

```
{
    "articles": [
        {
            "_id": 18683,
            "date_of_publication": "2015-01-20 18:01:xx",
            "headline": "27 cases in Hyderabad, 1 death in Punjab",
            "main_text": "While an outbreak of swineful  has claimed one life in Punjab, as many as 27 people in Telanganas
                capital Hyderabad have tested positive for the H1N1 virus, and their treatment is underway at the Gandhi
                Hospital here, while 35 cases are under observation for suspected influenza.\nRead Full Article At :: http://
                www.ddinews.gov.in/National/National%20-%20Headlines/Pages/Swineflu27casesinHyderabad,1deathinPunjab.aspx",
            "reports": [
                {
                    "diseases": [
                        "influenza a/h1n1"
                    ],
                    "event_date": "2015-01-20 18:01:xx",
                    "locations": [
                        {
                            "coords": "31.147, 75.341",
                            "country": "India",
                            "location": "Punjab"
```

Above: Postman screenshot for the case where there is no location parameter



```
GET calmclams.appspot.com/disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:1
```

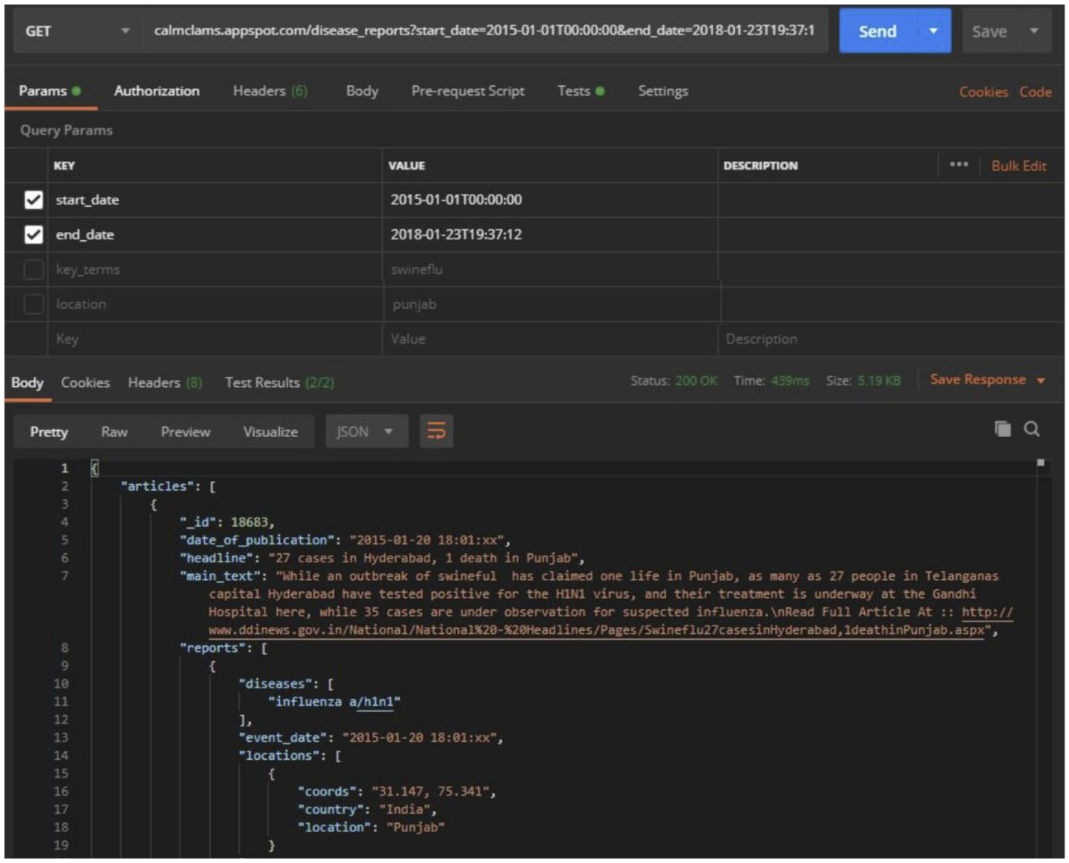
Query Params

KEY	VALUE	DESCRIPTION
✓ start_date	2015-01-01T00:00:00	
✓ end_date	2018-01-23T19:37:12	
key_terms	swineflu	
✓ location	punjab	

Status: 200 OK Time: 439ms Size: 5.19 KB

```
{
    "articles": [
        {
            "_id": 18683,
            "date_of_publication": "2015-01-20 18:01:xx",
            "headline": "27 cases in Hyderabad, 1 death in Punjab",
            "main_text": "While an outbreak of swineful  has claimed one life in Punjab, as many as 27 people in Telanganas
                capital Hyderabad have tested positive for the H1N1 virus, and their treatment is underway at the Gandhi
                Hospital here, while 35 cases are under observation for suspected influenza.\nRead Full Article At :: http://
                www.ddinews.gov.in/National/National%20-%20Headlines/Pages/Swineflu27casesinHyderabad,1deathinPunjab.aspx",
            "reports": [
                {
                    "diseases": [
                        "influenza a/h1n1"
                    ],
                    "event_date": "2015-01-20 18:01:xx",
                    "locations": [
                        {
                            "coords": "31.147, 75.341",
                            "country": "India",
                            "location": "Punjab"
                        }
```

Above: Postman screenshot for the case where there is no key_terms parameter



GET calmclams.appspot.com/disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:1

Query Params

	KEY	VALUE	DESCRIPTION
✓	start_date	2015-01-01T00:00:00	
✓	end_date	2018-01-23T19:37:12	
	key_terms	swineflu	
	location	punjab	
	Key	Value	Description

Status: 200 OK Time: 439ms Size: 5.19 KB

```
{
    "articles": [
        {
            "_id": 18683,
            "date_of_publication": "2015-01-20 18:01:xx",
            "headline": "27 cases in Hyderabad, 1 death in Punjab",
            "main_text": "While an outbreak of swineful  has claimed one life in Punjab, as many as 27 people in Telanganas
                capital Hyderabad have tested positive for the H1N1 virus, and their treatment is underway at the Gandhi
                Hospital here, while 35 cases are under observation for suspected influenza.\nRead Full Article At :: http://
                www.ddinews.gov.in/National/National%20-%20Headlines/Pages/Swineflu27casesinHyderabad,1deathinPunjab.aspx",
            "reports": [
                {
                    "diseases": [
                        "influenza a/h1n1"
                    ],
                    "event_date": "2015-01-20 18:01:xx",
                    "locations": [
                        {
                            "coords": "31.147, 75.341",
                            "country": "India",
                            "location": "Punjab"
                        }
```

Above: Postman screenshot for the case where there is no key_terms or location parameter

Additionally, we decided to make all queries case insensitive so that it would not restrict the returned data to users, especially since they can further filter the results as they desire.

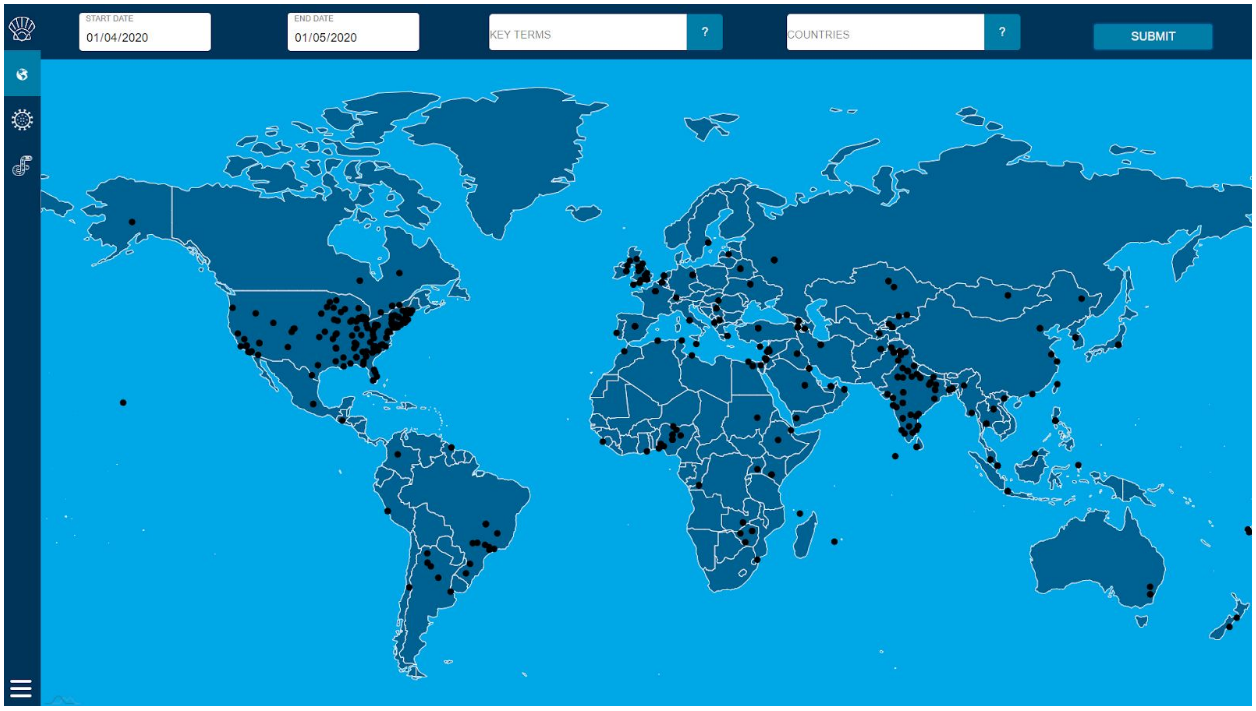
The *key_terms* parameter also accepts multiple values removing the need to submit a new query for each key term.

1.2 Analytics Platform (Web Application)

Our analytics platform targets researchers and analysts of past and ongoing epidemics. Our goal is to facilitate the comparison and correlation of both medical data sources and public responses to an epidemic in order to guide government policies in epidemic management. To accomplish this, we identified the following use cases we were required to implement for our system to be successful.

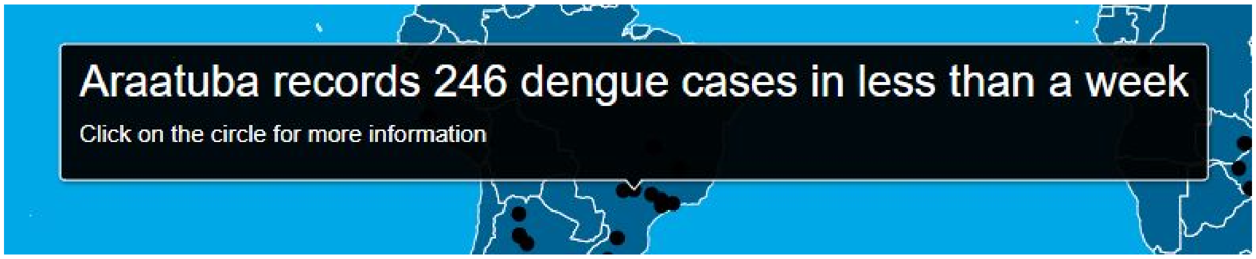
1.2.1 Monitor potential outbreaks

In order to quickly identify and prepare for potential outbreaks, a research team will want to view reports of outbreaks from across the globe. Our platform achieves this with a global map on which we can display the precise location of reports matching a given set of criteria.



Above: Global reports for the month of April 2020

By specifying a past date as their start date and the current date as the end date, users can filter down to recent reports in order to spot potential outbreaks of new or existing diseases. They can then quickly get the gist of each report by hovering over the mark on the map.

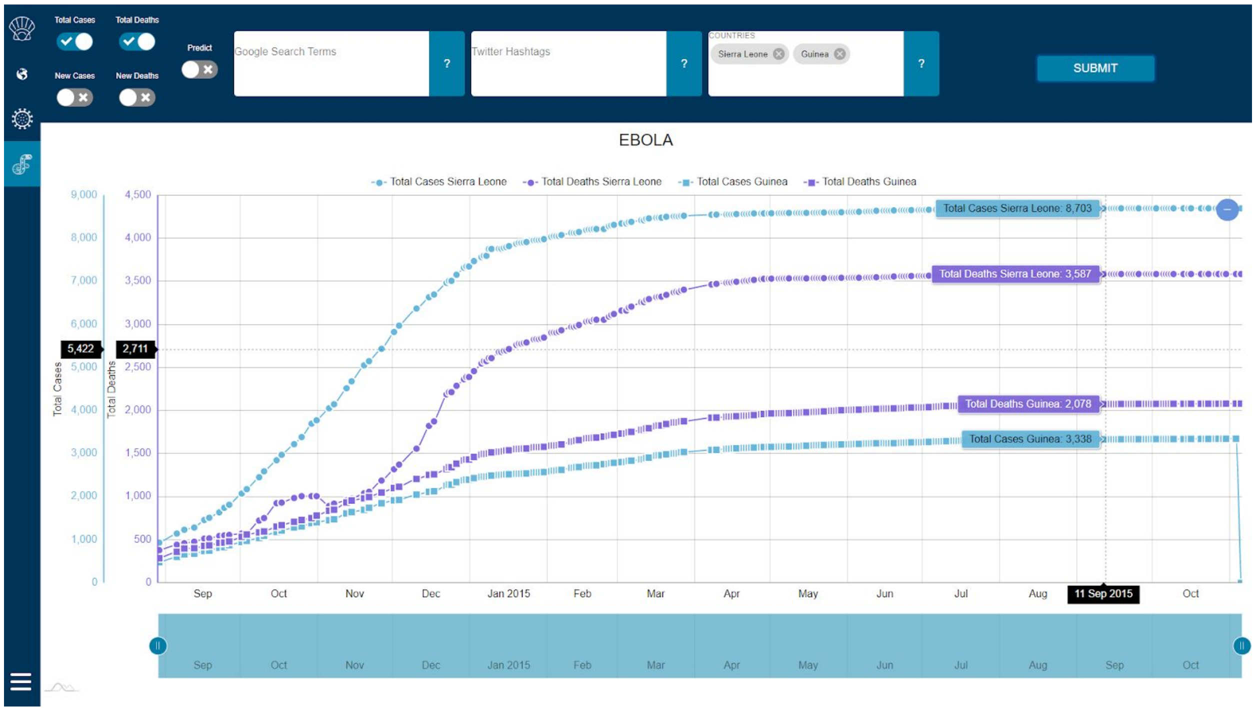

Araatuba records 246 dengue cases in less than a week

Click on the circle for more information

Above: The tooltip for a report of dengue in Brazil

1.2.2 View historical data

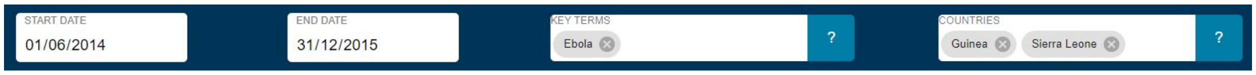
To prepare a response to a potential outbreak, researchers will need to be able to analyse the case data and responses of various countries in past outbreaks. They can compare case data using the graph page for a given disease. By selecting the types of data and the countries they want to view, researchers can quickly compare the effectiveness of different governments' responses in managing an epidemic.


Total Cases ✓ Total Deaths ✓ Predict ✗ New Cases ✗ New Deaths ✗

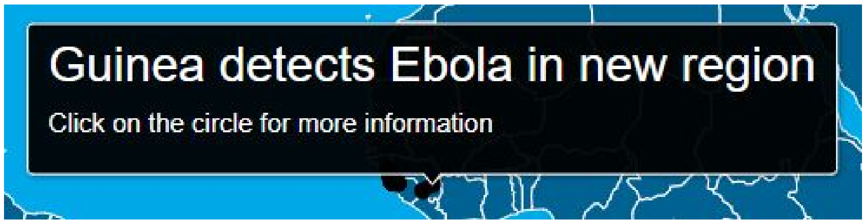
Google Search Terms | Twitter Hashtags | COUNTRIES: Sierra Leone, Guinea

Above: The total case and death data for Sierra Leone and Guinea for the 2014-2015 Ebola outbreak

They can then use the map page to search for reports for the disease from the countries they are comparing by entering the disease as a key term and the countries they're evaluating in the countries field in the search bar at the top of the page.


START DATE
01/06/2014
END DATE
31/12/2015
KEY TERMS
Ebola
COUNTRIES
Guinea
Sierra Leone

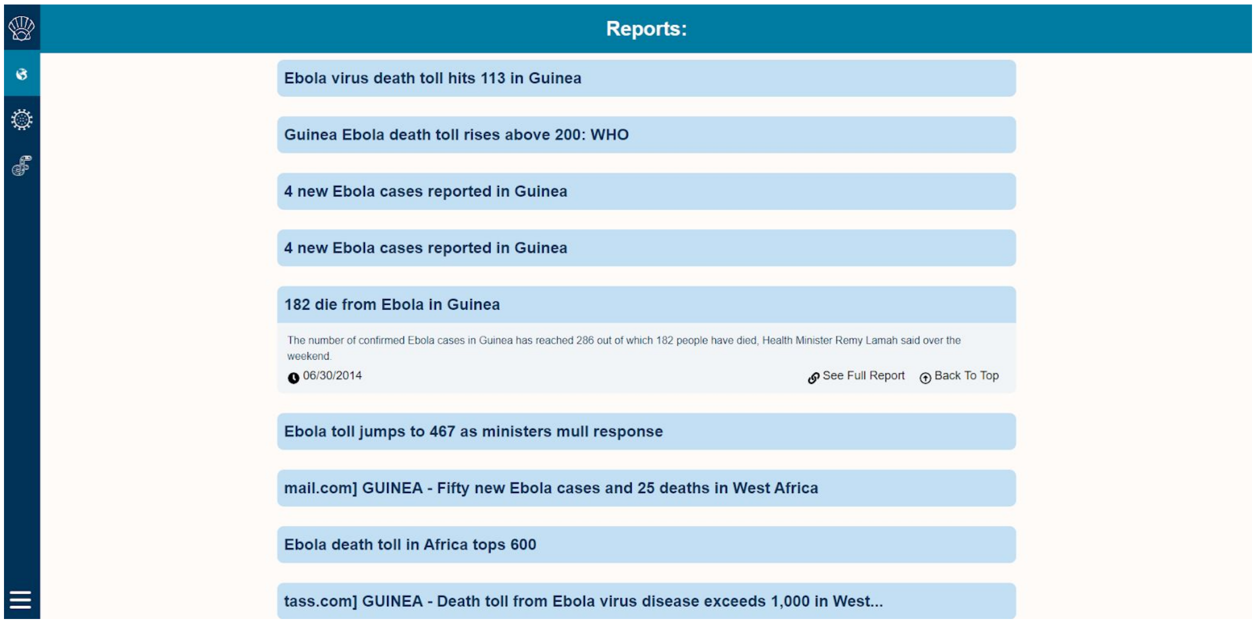
Above: Search criteria to show Ebola reports from Guinea and Sierra Leone from June 2014 to December 2015


Guinea detects Ebola in new region
Click on the circle for more information

Left: Ebola reports in Guinea and Sierra Leone from the above search

1.2.3 View detailed reports

When a search reveals reports of interest to researchers, they will want to be able to view the full details of the reports to get further information. They can do this by either scrolling down to below the map, where they will see a full list of expandible reports for their search, or clicking on a marker on the map, which will take them directly to the expansion of that report. From this expansion, they can click a link to navigate to the full report, or return to the map to continue searching.


Reports:

Ebola virus death toll hits 113 in Guinea

Guinea Ebola death toll rises above 200: WHO

4 new Ebola cases reported in Guinea

4 new Ebola cases reported in Guinea

182 die from Ebola in Guinea

The number of confirmed Ebola cases in Guinea has reached 286 out of which 182 people have died, Health Minister Remy Lamah said over the weekend.

06/30/2014 See Full Report Back To Top

Ebola toll jumps to 467 as ministers mull response

mail.com] GUINEA - Fifty new Ebola cases and 25 deaths in West Africa

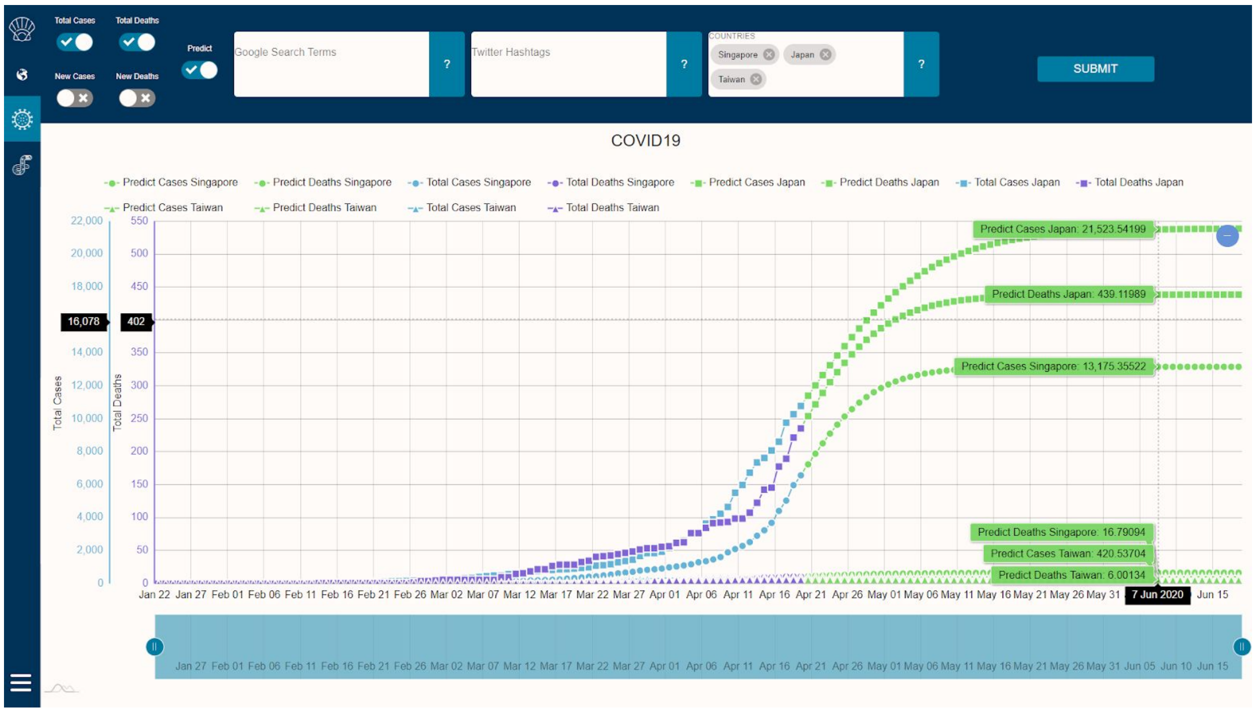
Ebola death toll in Africa tops 600

tass.com] GUINEA - Death toll from Ebola virus disease exceeds 1,000 in West...

Above: Expansion of a report from the search in section 1.2.2

1.2.4 Track current outbreaks and predicted trends

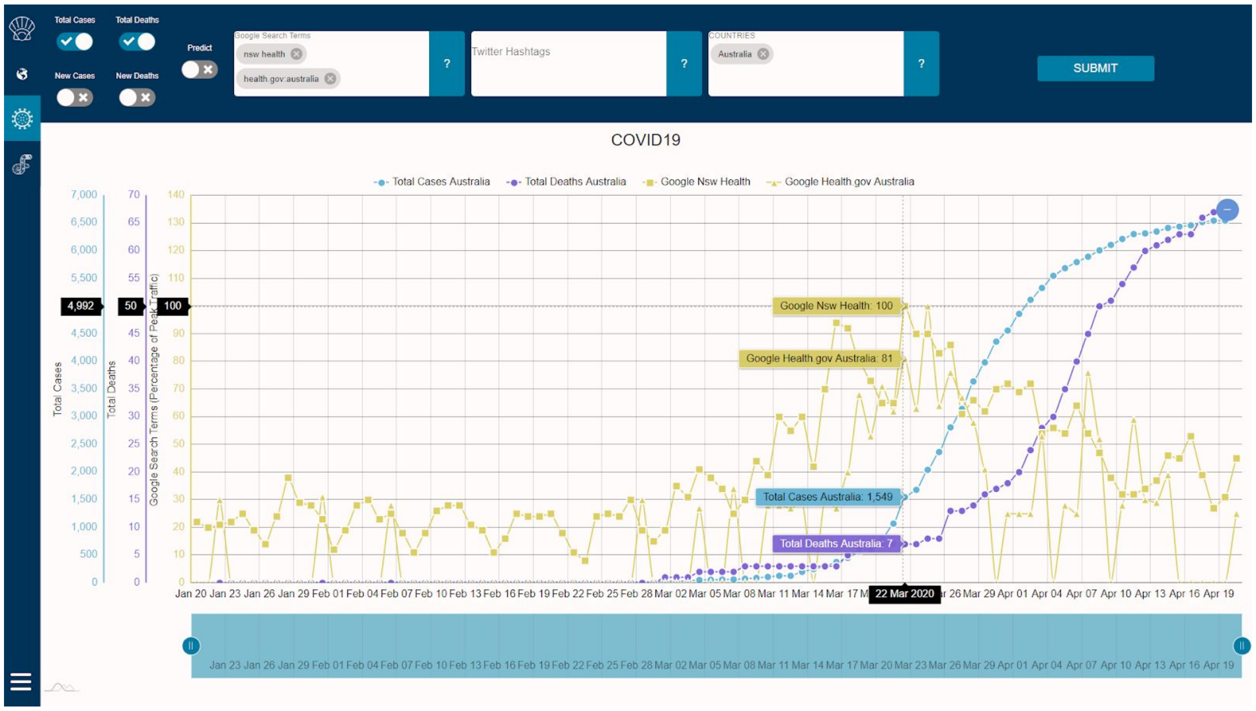
During an outbreak, researchers will need to be able to view current data and predictions on how the outbreak will develop. As with historical outbreaks, our analytics platform allows them to do this using the graph page for the disease. Countries can be compared by adding them to the countries selection, and predictions made for the number of cases and deaths by enabling the prediction option in the search bar. The prediction algorithm uses a custom logistic curve-fitting algorithm described in section 2.2.2, and can extrapolate current data arbitrarily far into the future.



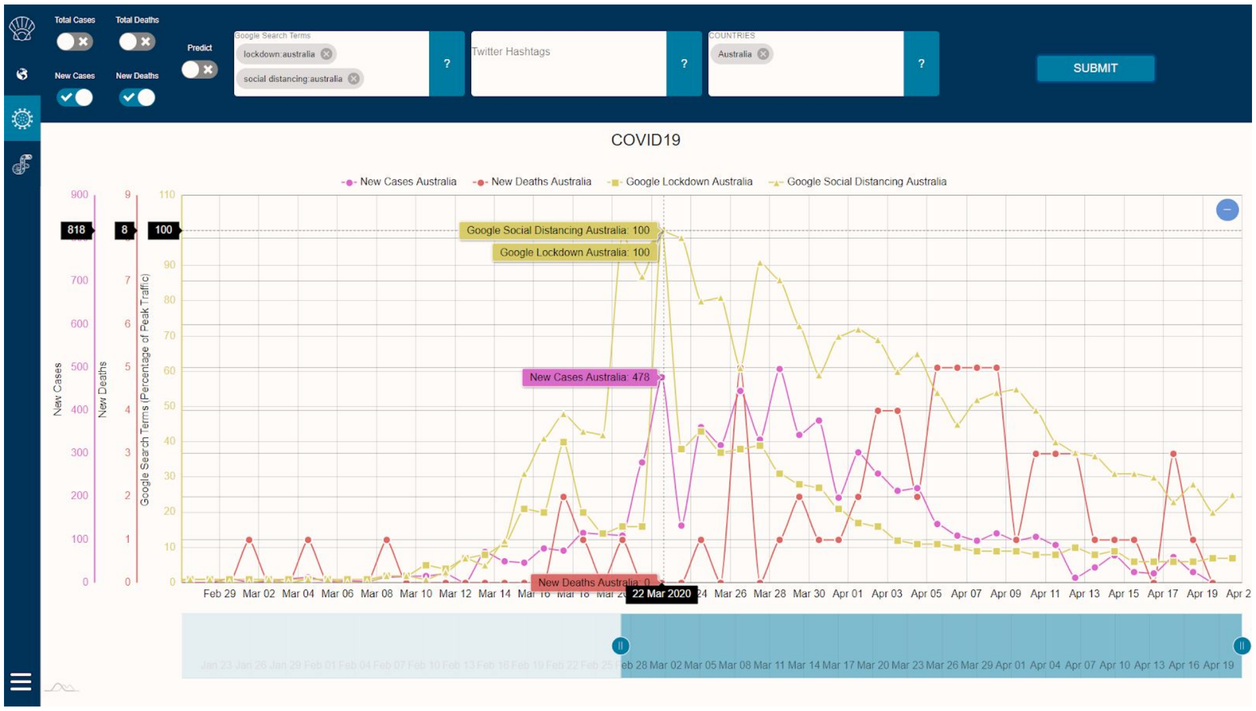
Above: A comparison of Singapore, Japan and Taiwan's data and predicted curves for COVID-19 in late April 2020

1.2.5 Compare case data to population behaviour

Finally, in order to guide government policies and actions during an ongoing epidemic, researchers need to be able to correlate case data with population behaviour. This can also be done using the graph page for the disease, by adding terms to the search bar. Researchers might use this to track public interest in health information over the course of the outbreak to determine when the government needs to remind the population to take precautions to limit the spread of the disease, or keep track of the amount of discourse regarding restrictions put in place during the epidemic.



Above: Australia's searches for health websites spike in the early stages of the COVID-19 pandemic before gradually decreasing back to normal levels



Above: Interest in social distancing and lockdown mirror Australia's new case numbers

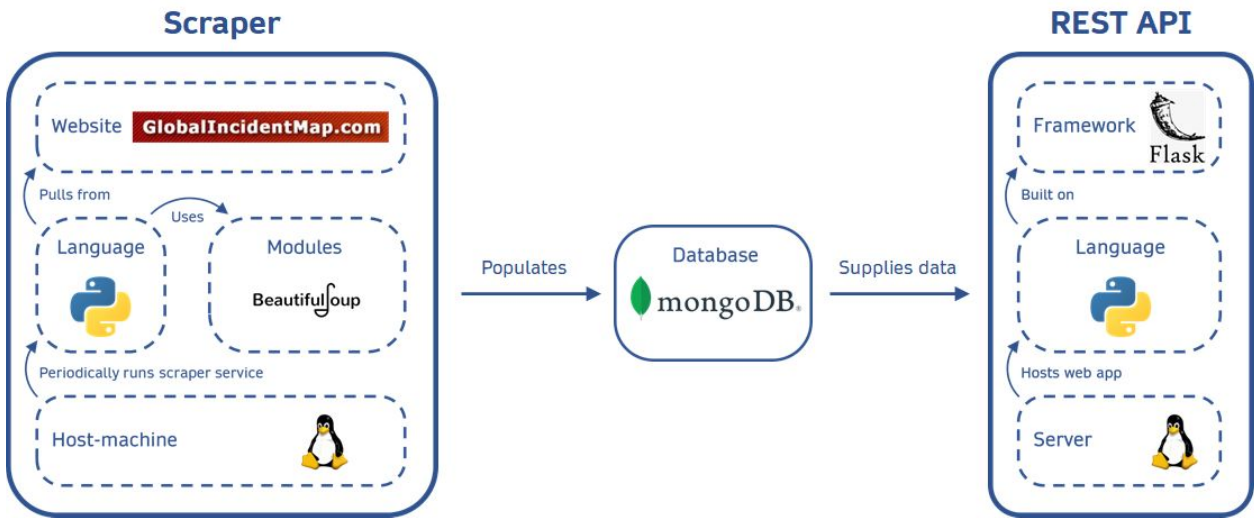
2 System Design and Implementation

2.1 Final Software Architecture

2.1.1 Web Scraper

For the web scraper we chose to use Python due to its ease of use. Being an interpreted language, developing a webscaper makes it quick and simple. The primary module we used was BeautifulSoup which allowed us to parse and search html pages with ease. The scraper primarily uses regex to search for keywords.

The site we scraped (globalincidentmap) offers detailed reports sorted in chronological order. Consequently the scraper running on the Google Cloud Platform is set up to scrape only on data it had not seen before hourly. In some cases there was more information to be found in the articles the reports links to. Consequently the scraper explores those articles as well when searching for keywords.



2.1.2 REST API

The API module was developed using Python with Flask, a web app development framework. This provided our single endpoint which allowed users to submit data using a HTTP request. Each request included a parameter that specifies what data the response should include. The web application then uses that parameter to filter the records in the database and return the appropriate results.

The web scraper explained in 2.1.1, populates a MongoDB database which the API accesses for the data it sends as a response. MongoDB allowed us to store data in the JSON format, the same format used throughout the application, which made it easier to interact with. This web application is currently hosted on the Google Cloud Platform.

To aid with the use of our API by other teams we decided to use Swagger. Swagger allowed for clear documentation of our API, in an easy to read form. It also allowed for someone to create dummy calls to our API through its Try It Out feature. Thus, anyone would be able to see how the calls work and what form they should be in. This aids with understanding as they are easily able to see what the calls should look like, thus easily integrating our API into their web app.

2.1.3 Web App

2.1.3.1 Frontend

The web app involves both a geographical and time series visualisation, so it is important to choose a frontend tech stack that enables us to quickly create a working app, while at the same time allowing for complex component design that can show off our features.

For the geographical visualisation and graphing tool we chose to use the AMCharts library. We found that this library had the most intuitive, flexible and feature rich option. There was also an added benefit of using the same library for both the map and graphs which reduced the time spent learning allowing us to develop our product faster. AMCharts is also well documented with many examples, tutorials and API level documentation allowing anyone less familiar with the library to be caught up to speed quickly.

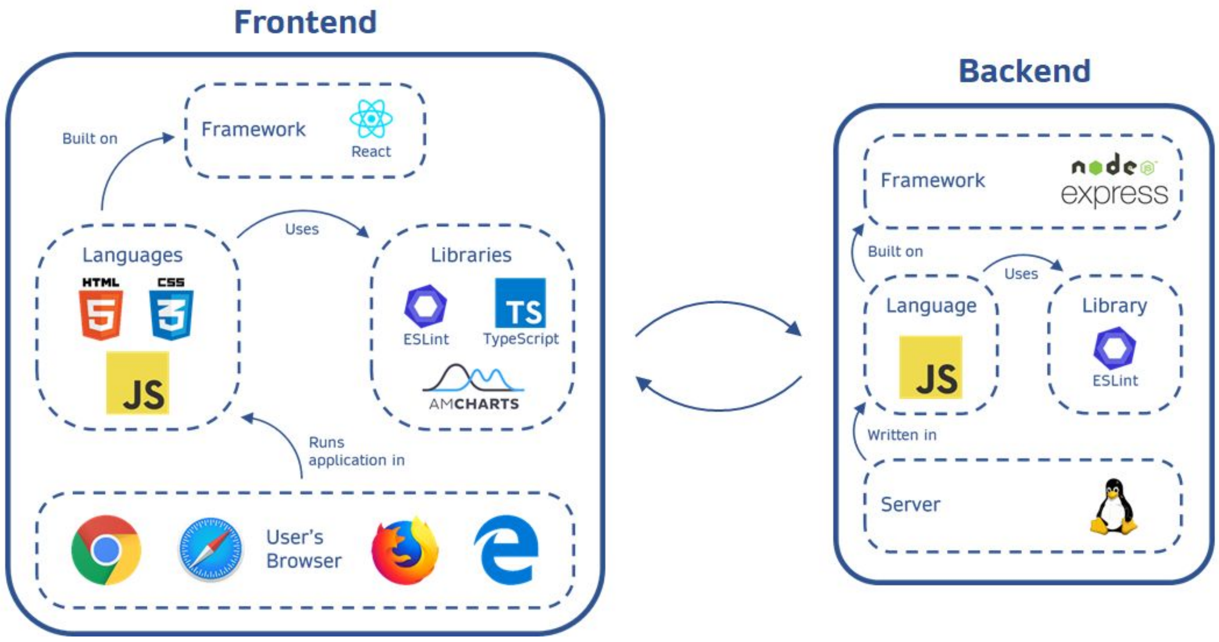
The framework we chose to use was React. This is for two reasons. Firstly, many of us have industry experience working with React, and so we can write the website at a faster pace. React also allows for higher complexity with its component system, enabling our team to go above and beyond, without being hindered by the framework.

There are two supporting libraries that we are utilising: Typescript and ESLint.
Typescript allows us to typecast our javascript code, essentially reducing the time spent tracking bugs caused by javascript’s dynamic typing system. ESLint allows us to standardise our code style, reducing the time it takes for new developers to understand what is going on. It also does this automatically, so no extra time is spent on formatting.

2.1.3.2 Backend

Our backend provides a single endpoint for all the data used by the graphs. This involves correlating data from multiple APIs (listed in the next section) and constructing predictions for total cases and deaths. This required a fast development environment which is able to perform simultaneous fetch requests and the complex mathematics needed for our prediction algorithm.

Consequently, we decided to use NodeJS Express, which is known for its flexibility and ease of use. Just like the frontend, we will use ESLint.

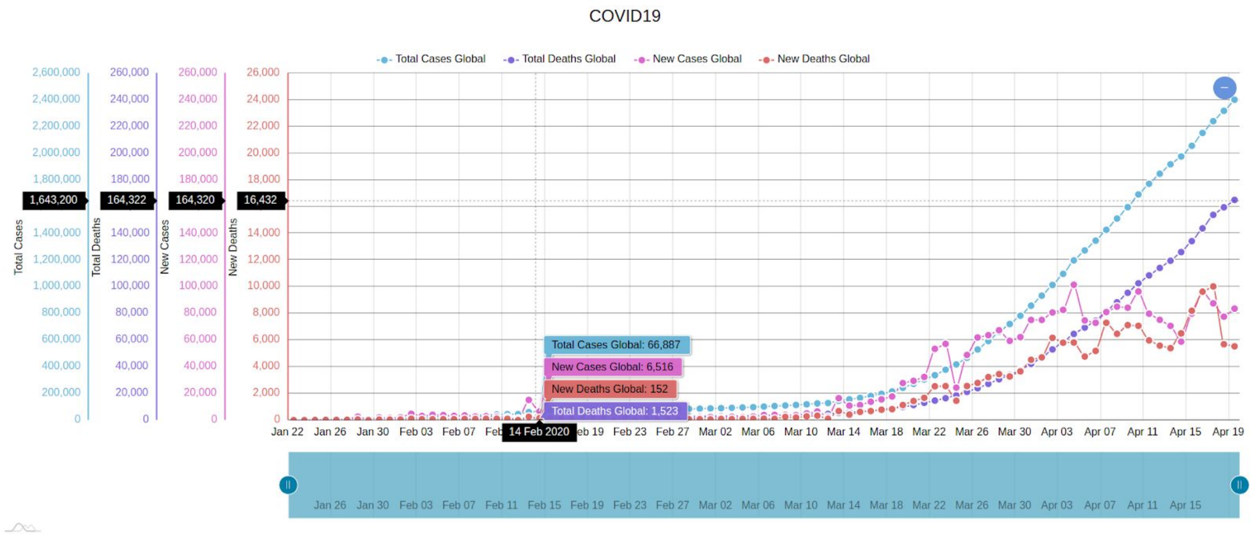


2.2 Additional Features

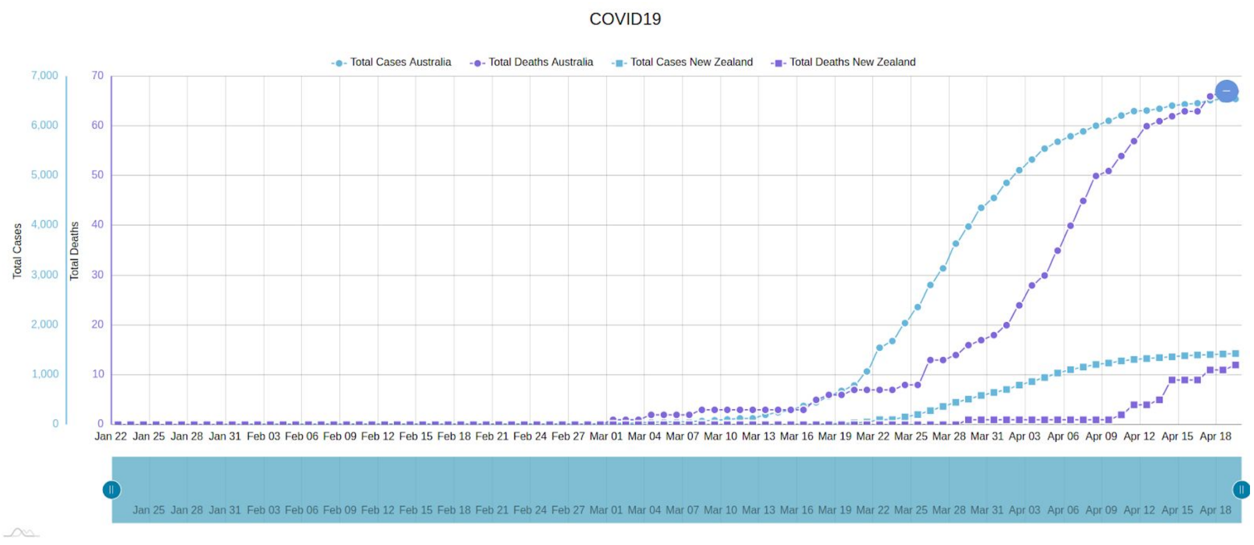
2.2.1 APIs

To generate the data that we needed for our graphing engine, we needed to interact with a variety of external APIs for data collection and transformation. These APIs can be split up into a series of sections; COVID-19, ebola, twitter and google trends.

To collate the COVID-19 data required, we interacted with two APIs. The first API used was https://covid19api.com/.This API was used to obtain data specific to individual countries, used in our country by country comparison. However, due to the structure of this API, it would have been difficult and require a large number of requests to generate global covid data. So we decided to use a different api, https://covid-api.com, which used data from Johns Hopkins to give day by day information on the global state of the epidemic. With these two APIs, our backend was able to collate all COVID-19 related data.

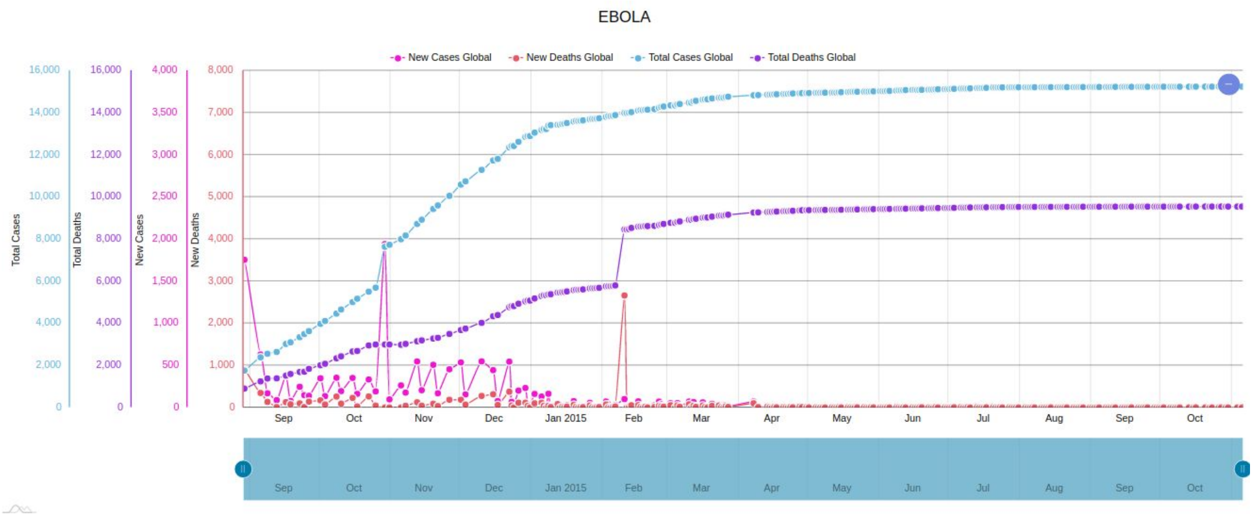


Above: The graph displaying global data for covid-19



Above: The graph displaying cumulative data for both Australia and New Zealand

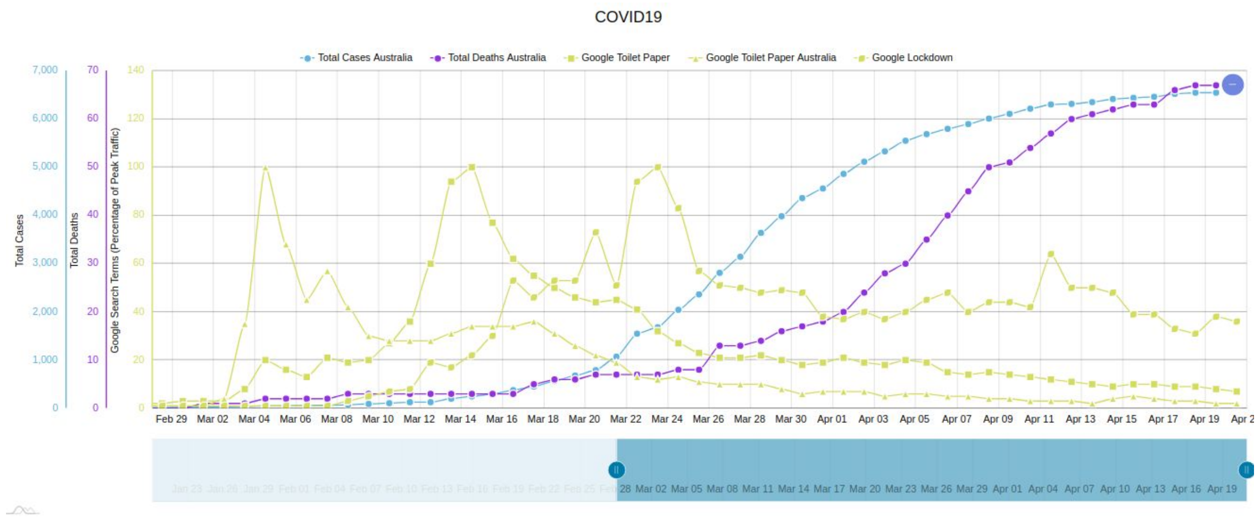
Whilst there was an abundance of APIs for COVID-19, due to it being a current global pandemic, finding data for Ebola was harder. We ended up using a csv obtained from data.world, which we used our backend to parse and serve up data from. This csv contained all the relevant information needed for the graphing engine. However, as this data was older and the outbreak of ebola was smaller, the data collected was not as effective.



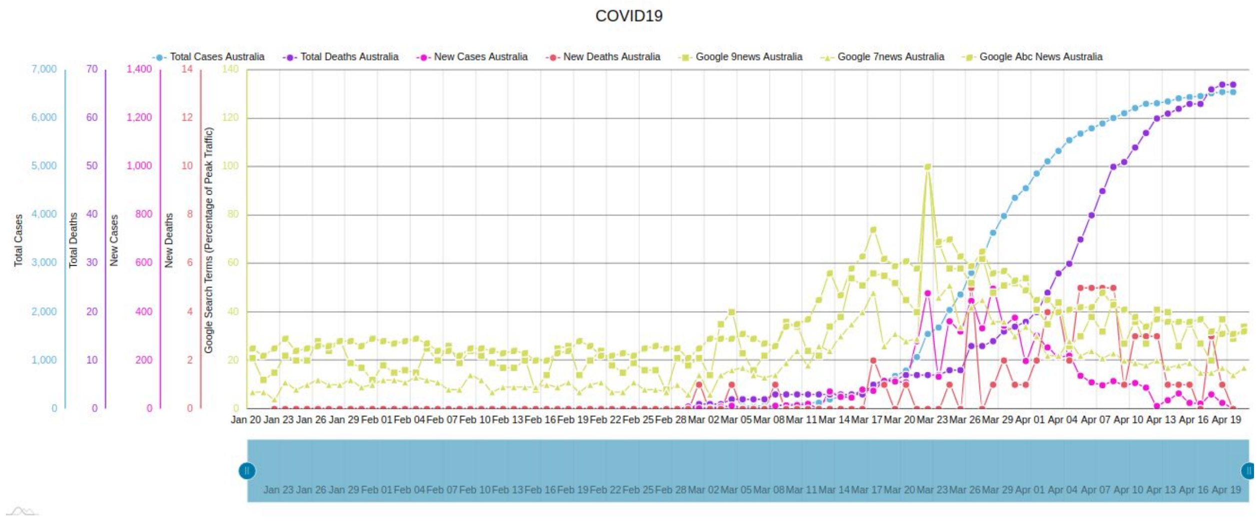
Above: The graph displaying the global data for the 2014-2015 West Africa Ebola Outbreak

When we attempted to integrate twitter trend information into our backend, we ran into an issue. We required an enterprise account to use the data, and that took time to have authorized by twitter, so we were unable to integrate it for the demonstration. If our website was to go into production, we would ensure to integrate the twitter API into our backend, to get a better understanding of public response.

However, we were able to integrate and use to great effect the Google Trends Data. This was done by using the google-trends-api module in nodejs, which acted as an intermediary API between our backend and the Google Trends APIs. This simplified the integration process, and allowed us to quickly and easily include data about the public's response to an outbreak through what they are searching for. Also, the google-trends-api module allows for searching by country, allowing us to find data that is region specific and compare the responses of different countries. Below are some examples of the data we were able to collect:



Above: The graph displaying global covid data against searches for toilet paper, first in australia then globally, and searches for lockdown.

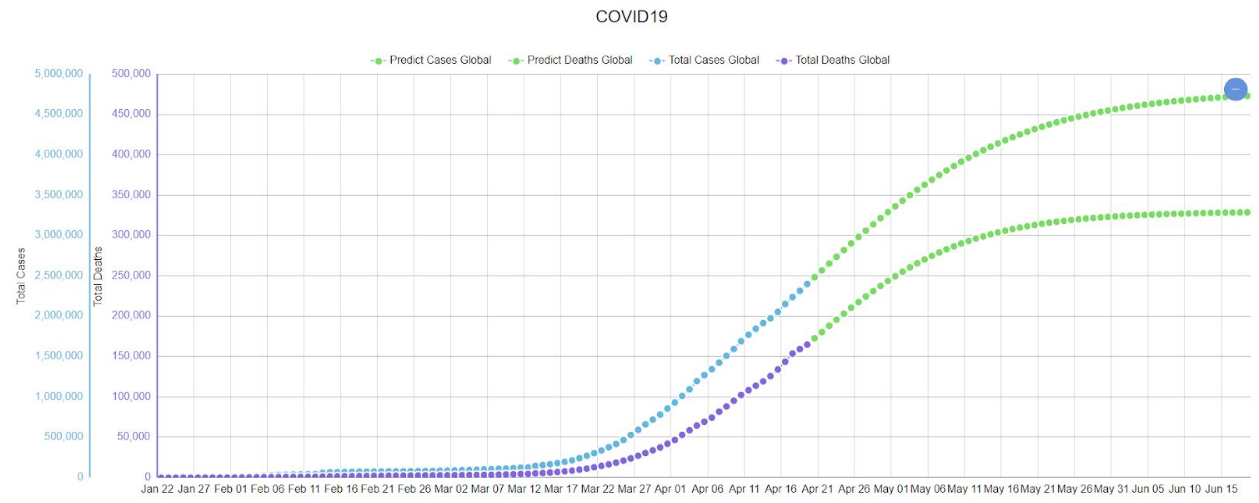


Above: The graph displaying the percentage of google search traffic for 9news, ABC news and 7news compared to Australia's total cases and deaths (Notice that they all peak on Australia's first peak in new cases)

All of these APIs helped enhance our web application by providing useful, realtime information to the user that allows them to make educated decisions surrounding the covid app.

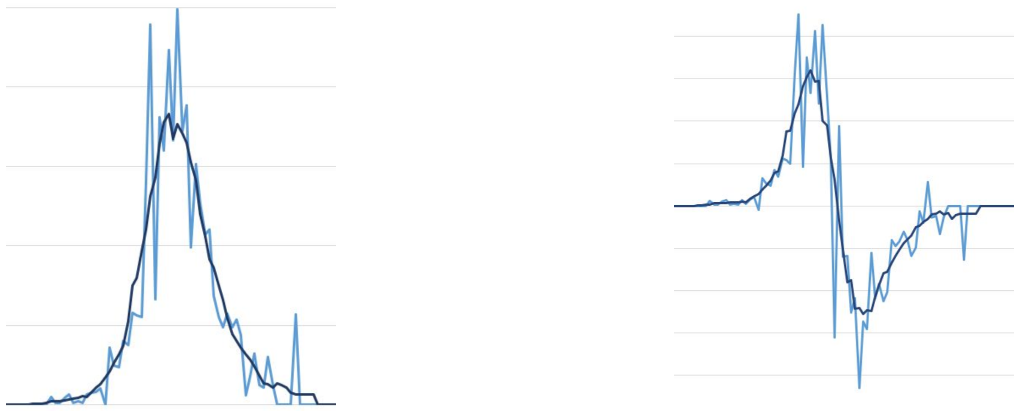
2.2.2 Algorithms

The analytics platform predicts future values for all case and death datasets using a custom logistic curve-fitting algorithm. The algorithm determines the most likely point of inflection on the curve in order to create an equation for a logistic curve that best fits the data, which it can use to estimate values arbitrarily many days into the future.



Above: The prediction algorithm applied to global cases (blue) and deaths (purple) for COVID-19 to estimate values for the next 2 months (green)

To determine the point of inflection, we first calculate the 1st derivative at each existing data point. Typically this data is "spiky", as real-world values don't precisely follow a mathematical equation, so we dampen it by taking the average of the derivatives at the nearest 9 points. We then use the dampened 1st derivative values to calculate the 2nd derivative, which we again dampen using the nearest points. For a set of data which has reached the top of the logistic curve, this results in the 1st and 2nd derivative curves below:



Left: Raw 1st derivative (light) and dampened 1st derivative (dark)
Right: Raw 2nd derivative (light) and dampened 2nd derivative (dark)

Of course, if we were always given a completed logistic curve, there'd be no purpose in predictions. Typically a dataset will give us part of the beginning of the curve, so we use the two derivatives to determine the likely location and gradient of the point of inflection, where the rate of infection hits a maximum before beginning to decrease. From these three values, we calculate the curve as follows:

Let (x_i, y_i) be the coordinate of the point of inflection, and m_i be the gradient of the curve at (x_i, y_i).

The equation of a logistic curve is

$$f(x) = \frac{L}{1 + e^{-k(x-s)}}$$

where

L is the height of the curve

s is the x-value of the centre of the curve

k is a constant that determines the "steepness" of the curve

A logistic curve has rotational symmetry around the point of inflection, so

$$L = 2y_i$$

$$s = x_i$$

which allows us to rewrite the equation of the curve as

$$f(x) = \frac{2y_i}{1 + e^{-k(x-x_i)}}$$

To determine k, we differentiate the original equation to get

$$f'(x) = \frac{2ky_i e^{-k(x-x_i)}}{(1 + e^{-k(x-x_i)})^2}$$

We know the gradient m_i at x_i, the point of inflection, so

$$\begin{aligned} m_i &= f'(x_i) \\ &= \frac{2ky_i e^{-k(x_i-x_i)}}{(1 + e^{-k(x_i-x_i)})^2} \\ &= \frac{ky_i}{2} \end{aligned}$$

Rearranging gives

$$k = \frac{2m_i}{y_i}$$

So the equation of our curve is

$$f(x) = \frac{2y_i}{1 + e^{-\frac{2m_i}{y_i}(x-x_i)}}$$

Finally, we offset this equation on the y-axis to align with the final real data point to account for any real-world variance. We can then apply the resulting equation to any date to estimate the number of cases or deaths on that date.

2.3 Key Benefits/Achievements

One great benefit that our web application produces is the consolidation of multiple news and data sources into a single source of information. Our website has integrated with multiple data sources and APIs and combined that data into a single data source that can be accessed through the report search functionality, as well as the graph engine.

We also achieved great visibility in this data through the use of our map and graph features. These forms of displaying data, as opposed to simply listing the reports, provides more visibility to the user and makes digesting the data that we have collected far easier. This visibility aids the user with the benefit of being able to make meaningful comparisons and observations faster, aiding in decision making for times of an outbreak and potential outbreak potential.

To aid with the user experience of our web application we made sure to develop a responsive backend that is able to deliver the information that the user asks for within a reasonable time. We did this by making multiple requests to data sources simultaneously, reducing the overall response time to roughly equivalent to one request to an external data source. This responsiveness enables a user to make more searches in the same amount of time, meaning our web application provides more benefit to the end user.

We understood that the goal of our application should be to become an analysis platform for all outbreaks, not just the current Covid-19 pandemic, so we made sure that all of our technology was universal and can be applied to a whole range of diseases. This was easily achieved in our API, as the data source we were using did not discriminate between diseases. However, this generality proved more difficult in our backend. This is because most data sources that had the granularity of data that we required focus only on one disease at a time. However, more diseases could be added into our system by easily integrating their data into the backend, which is something we would prioritize with continued development.

These are the key achievements of our web application and API that will enable us to benefit our end users.