Final Report

Calm Clams

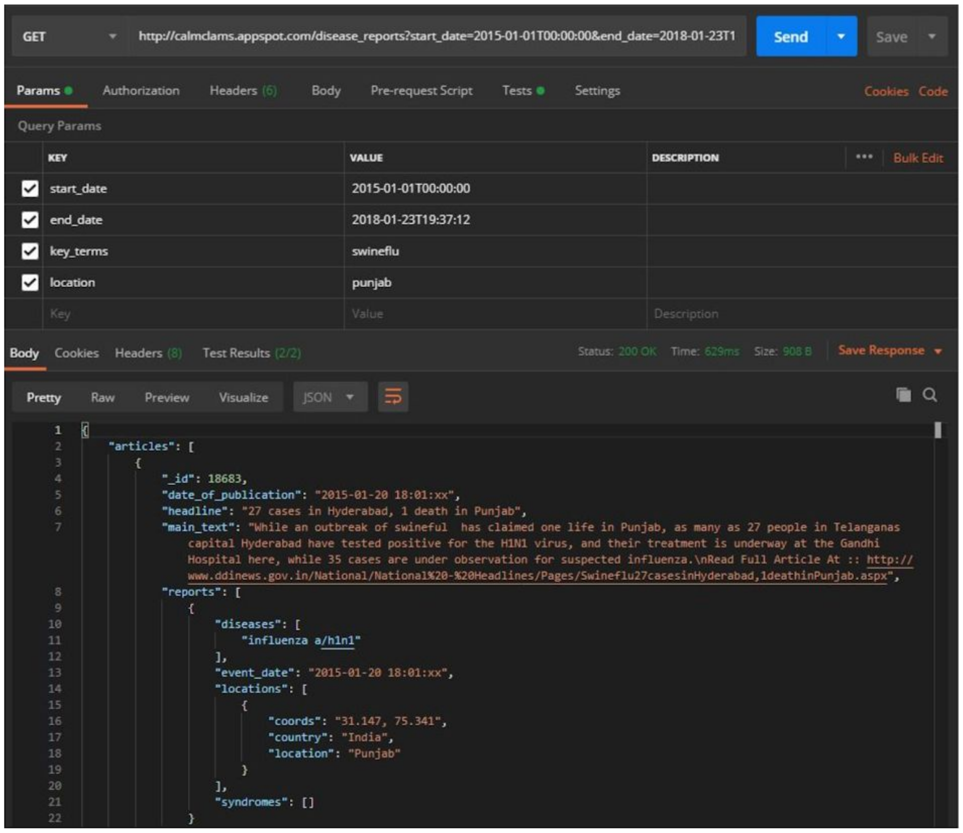
Carlin Williamson - z5122521
Adam Parslow - z5122506
Lucas Masina - z5204754
Marcus Majchrzak - z5208133
Sean Zammit - z5207033

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
```

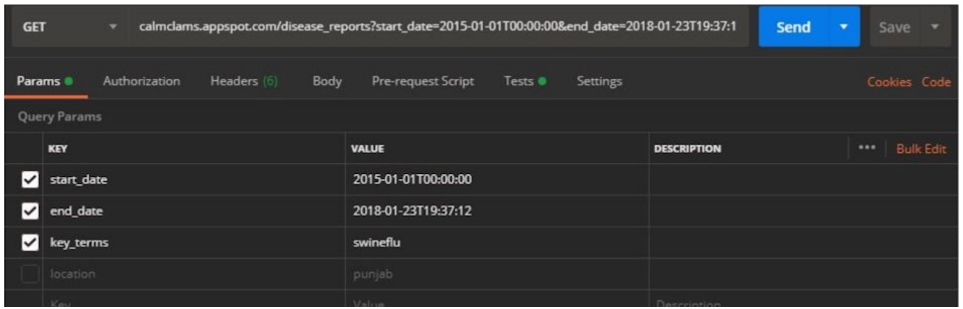
Query Params

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

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

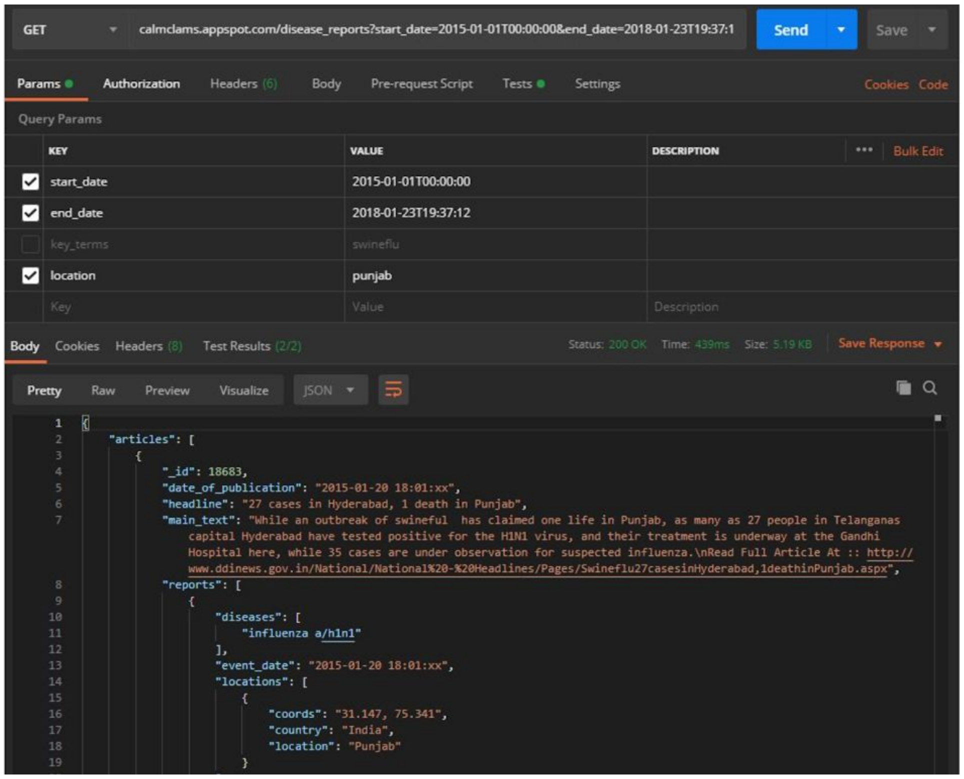


```
GET  calmclams.appspot.com/disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:1
```

Query Params

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

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

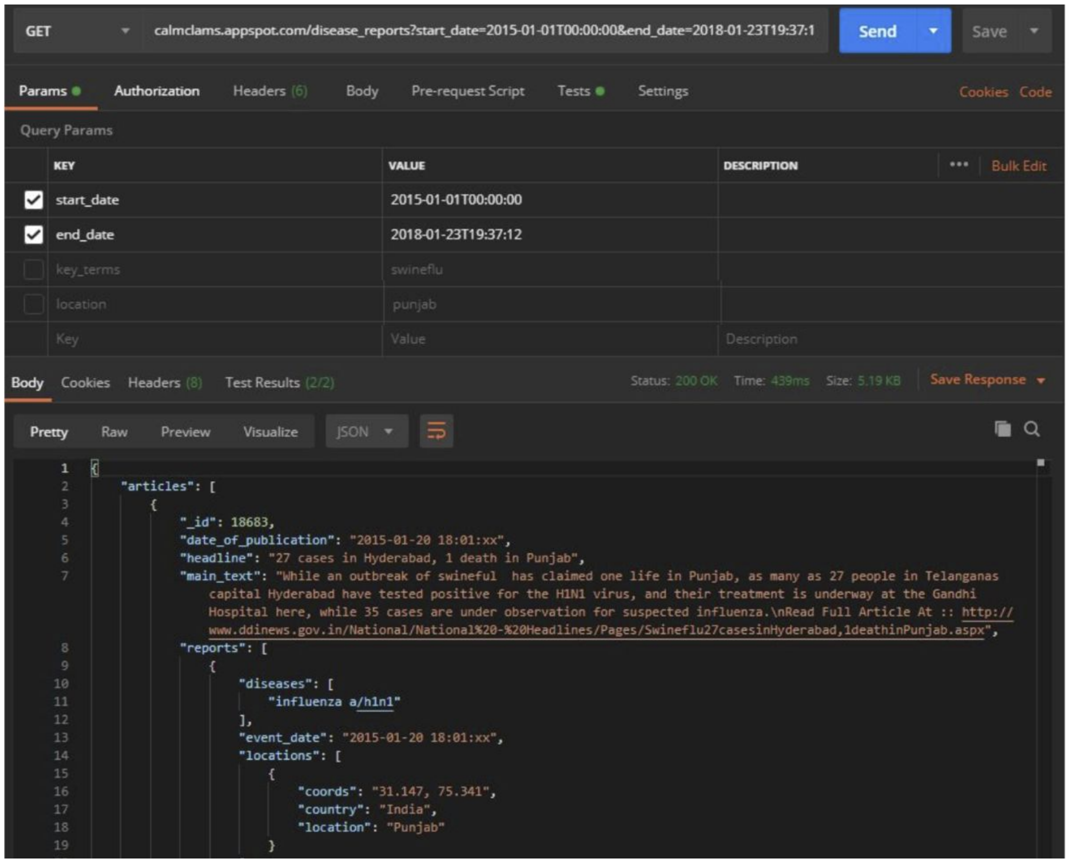


calmclams.appspot.com/disease_reports?start_date=2015-01-01T00:00:00&end_date=2018-01-23T19:37:1

	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

```
{
    "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



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

	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.

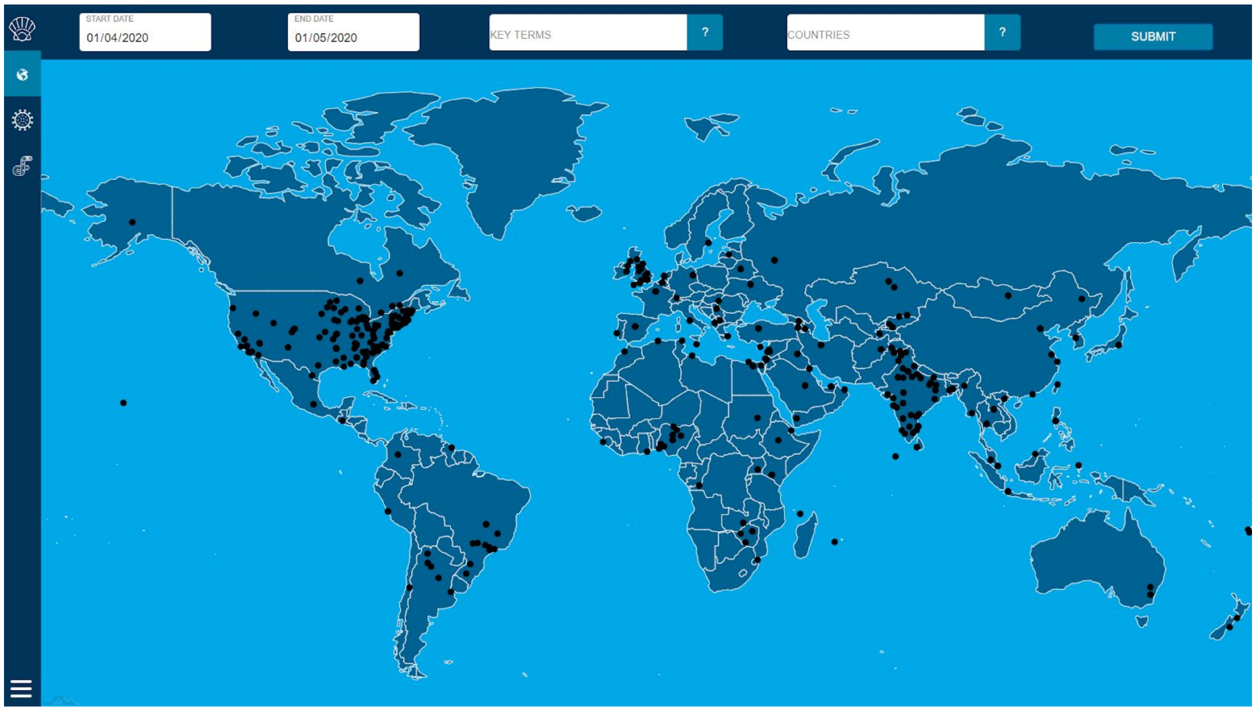
The *key_terms* parameter also accepts multiple values removing the need to submit a new query for each key term.

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.

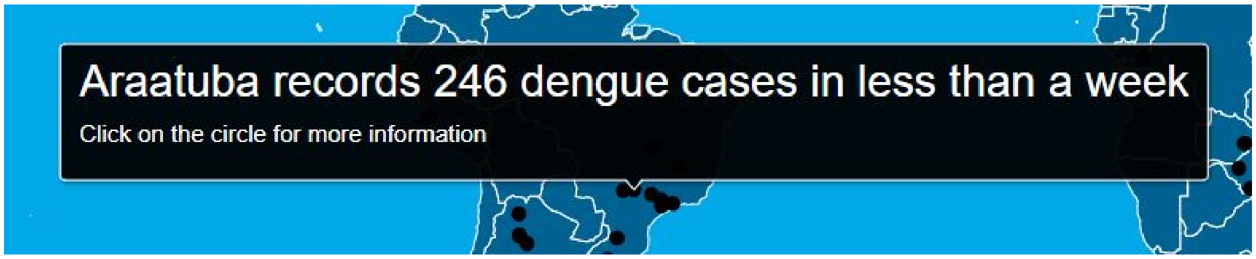
1.2.1 Monitor potential outbreaks

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.

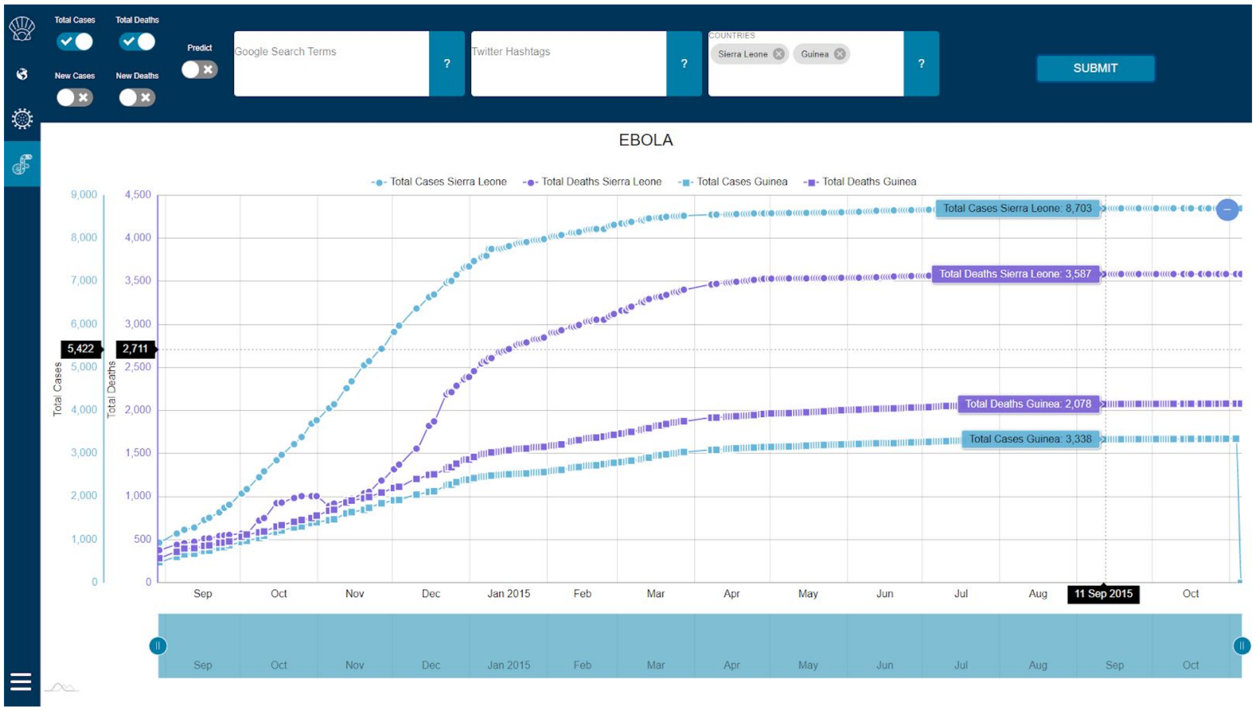

Araatuba records 246 dengue cases in less than a week

Click on the circle for more information

Above: The tooltip for a report of dengue in Brazil

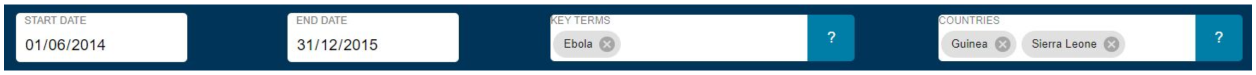
1.2.2 View historical data

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.

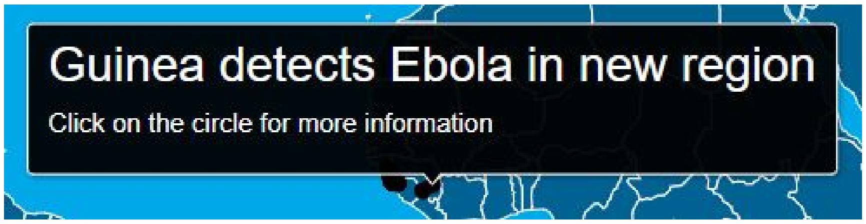


Above: The total case and death data for Sierra Leone and Guinea for the 2014-2015 Ebola outbreak

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.



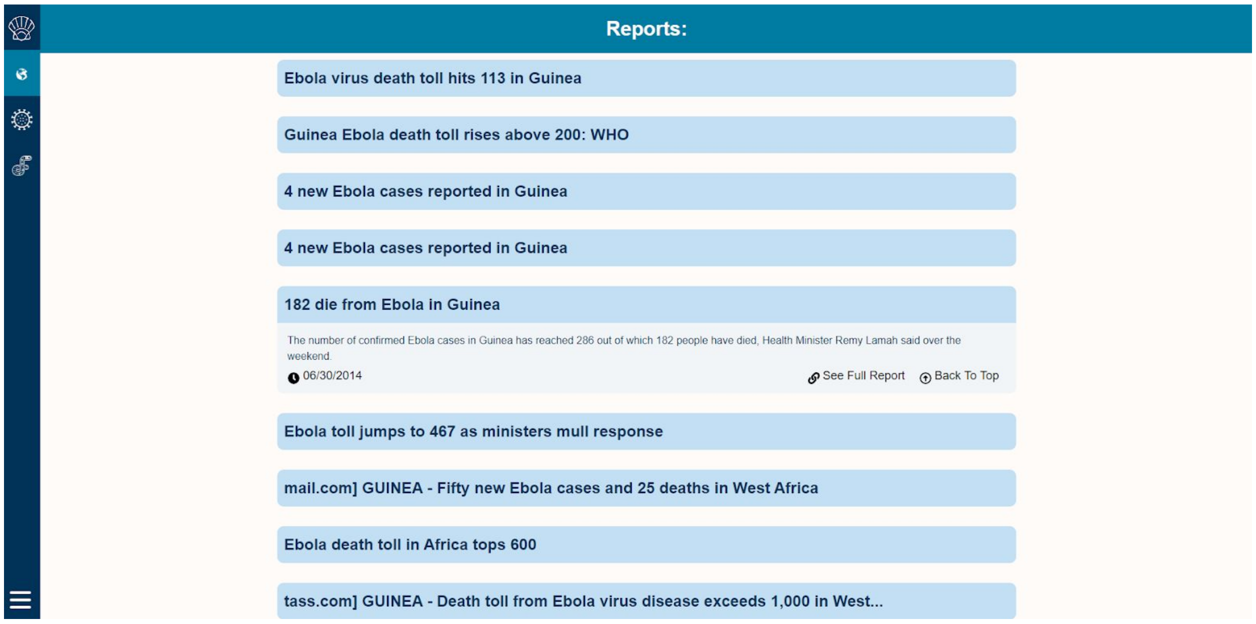
Above: Search criteria to show Ebola reports from Guinea and Sierra Leone from June 2014 to December 2015


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

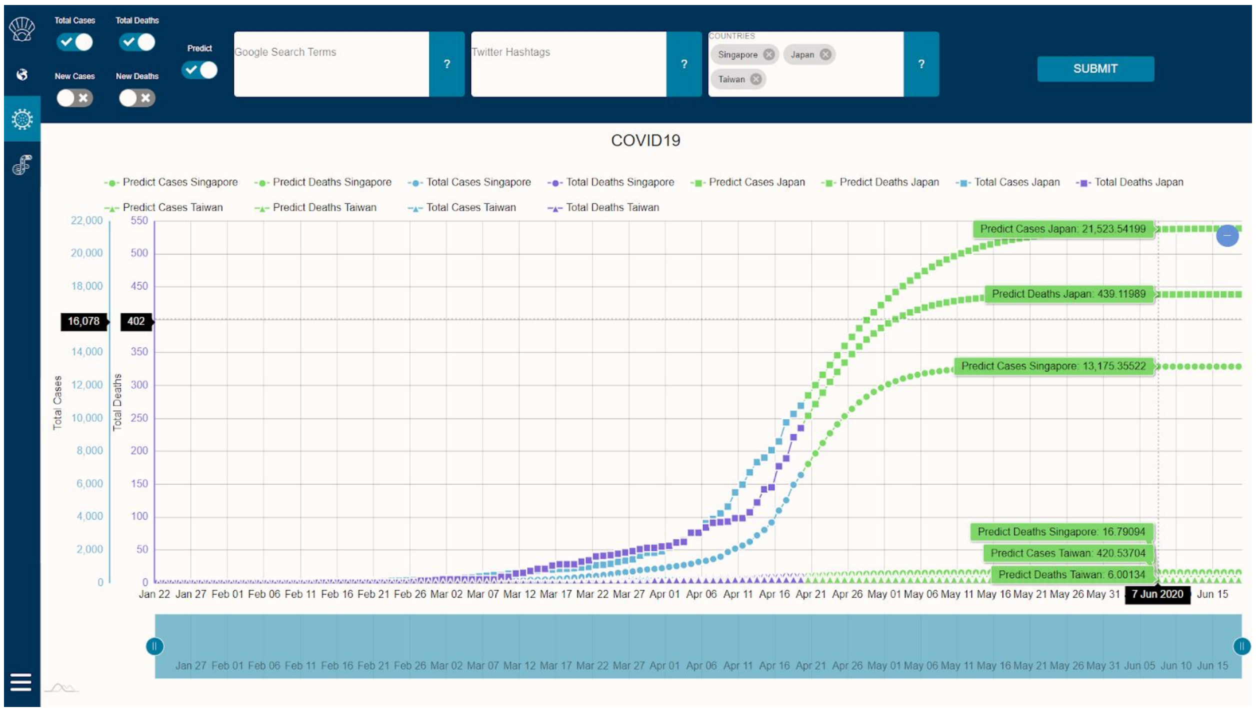
  The number of confirmed Ebola cases in Guinea has reached 286 out of which 182 people have died, Health Minister Remy Lamah said over the weekend.

  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

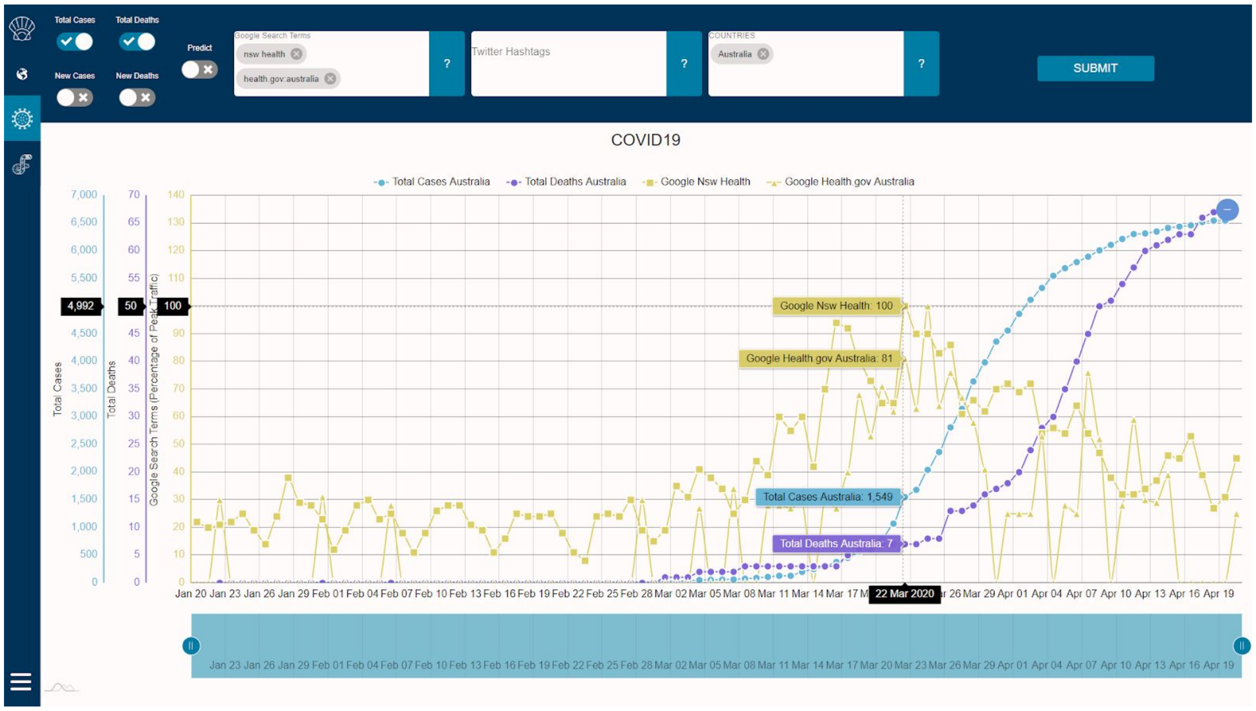
During an outbreak, researchers will need to be able to view current data and predictions on how the outbreak will develop. As with historical outbreaks, our analytics platform allows them to do this using the graph page for the disease. Countries can be compared by adding them to the countries selection, 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.



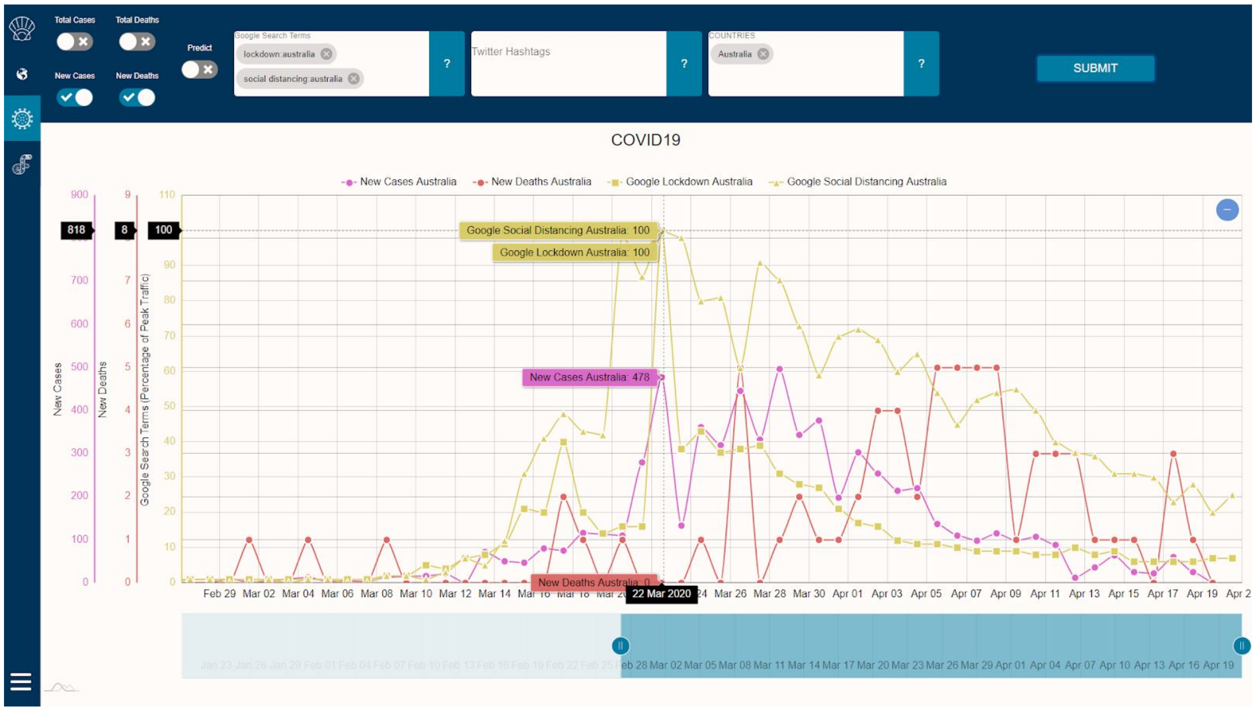
Above: A comparison of Singapore, Japan and Taiwan's data and predicted curves for COVID-19 in late April 2020

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

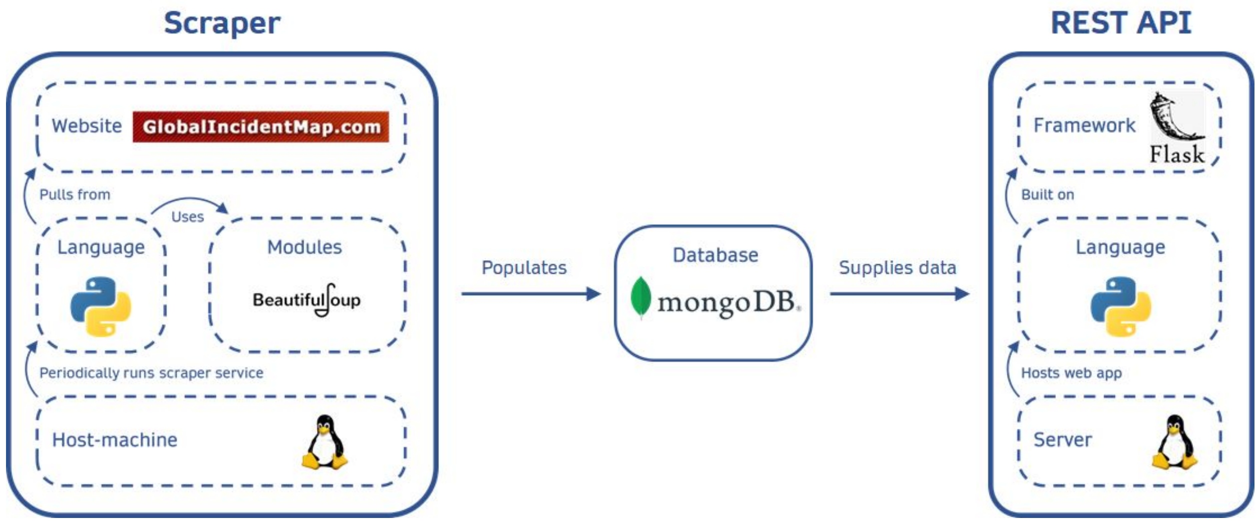
2 System Design and Implementation

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.

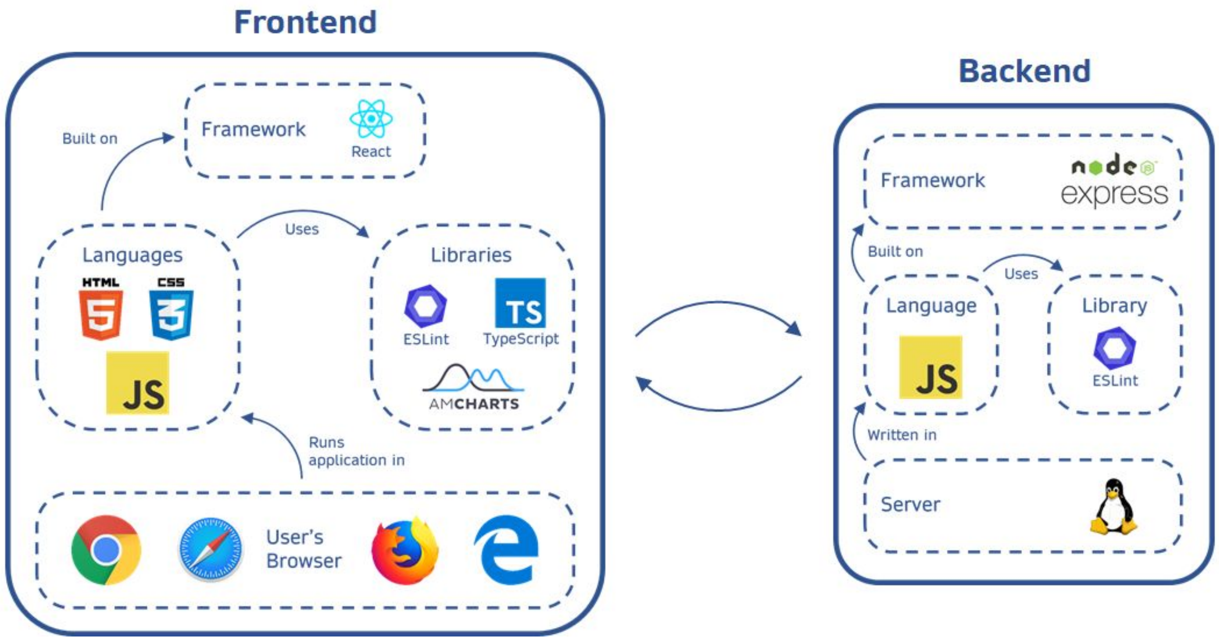
The framework we chose to use was React. This is for two reasons. Firstly, many of us have industry experience working with React, and so we can write the website at a faster pace. React also allows for higher complexity with its component system, enabling our team to go above and beyond, without being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

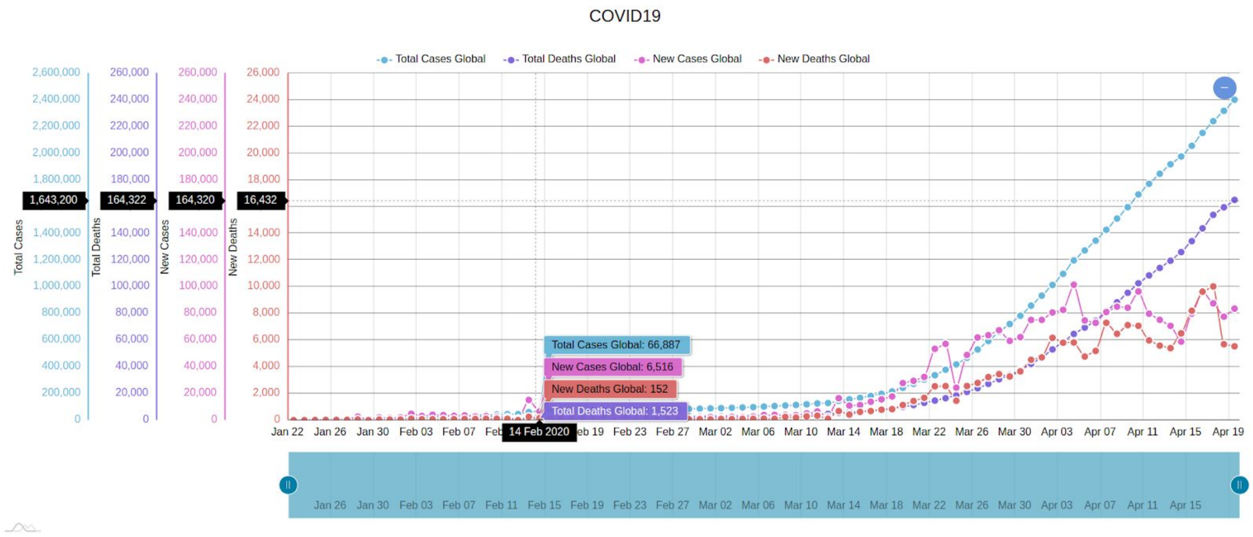


2.2 Additional Features

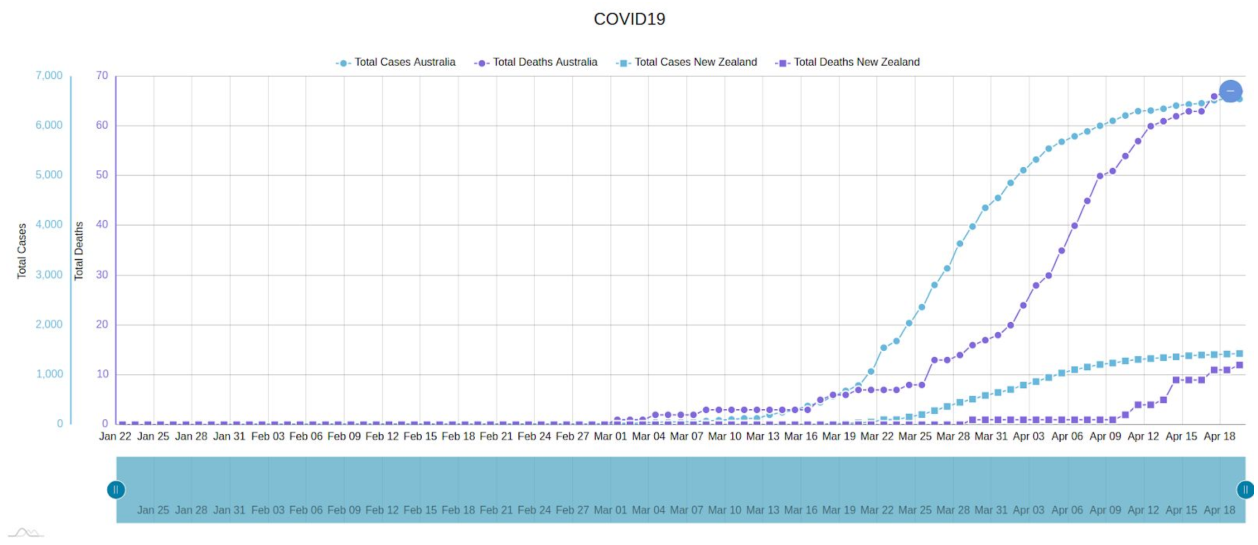
2.2.1 APIs

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.

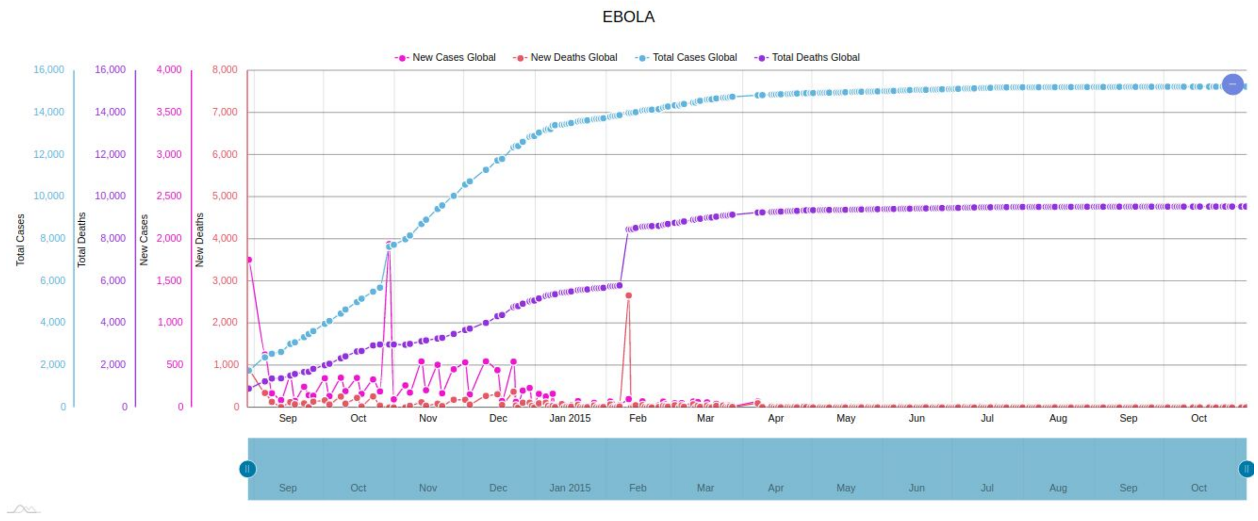


Above: The graph displaying global data for covid-19



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

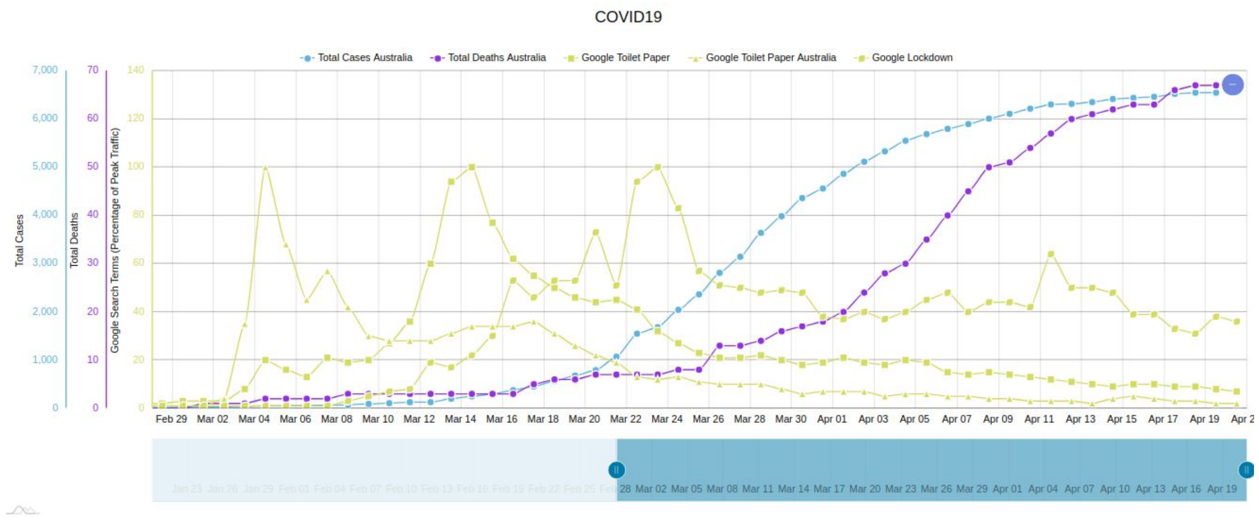
Whilst there was an abundance of APIs for COVID-19, due to it being a current global pandemic, finding data for Ebola was harder. We ended up using a csv obtained from data.world, which we used our backend to parse and serve up data from. This csv contained all the relevant information needed for the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



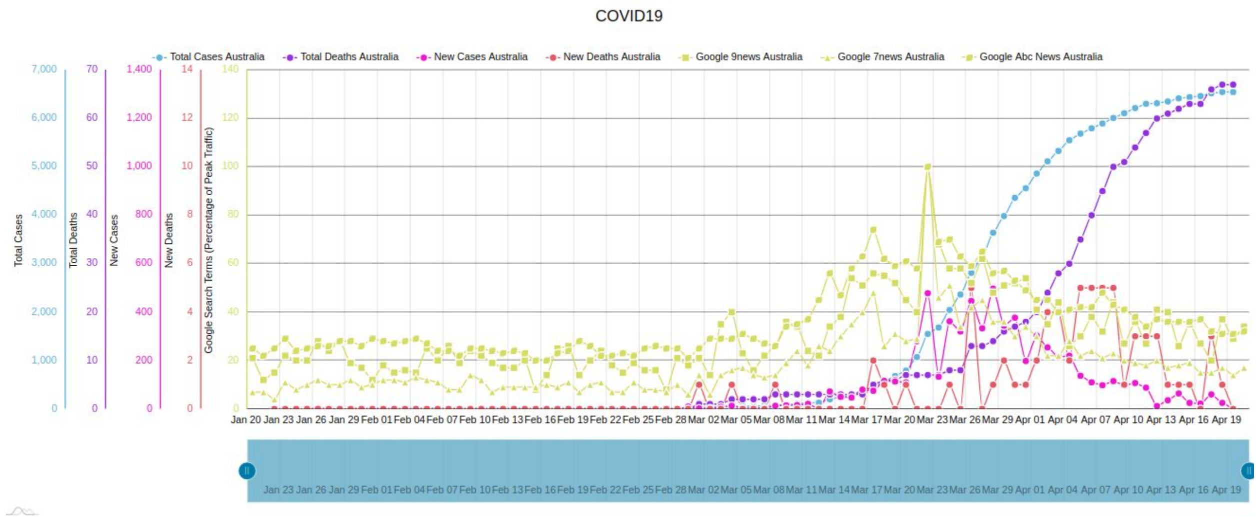
Above: The graph displaying the global data for the 2014-2015 West Africa Ebola Outbreak

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.

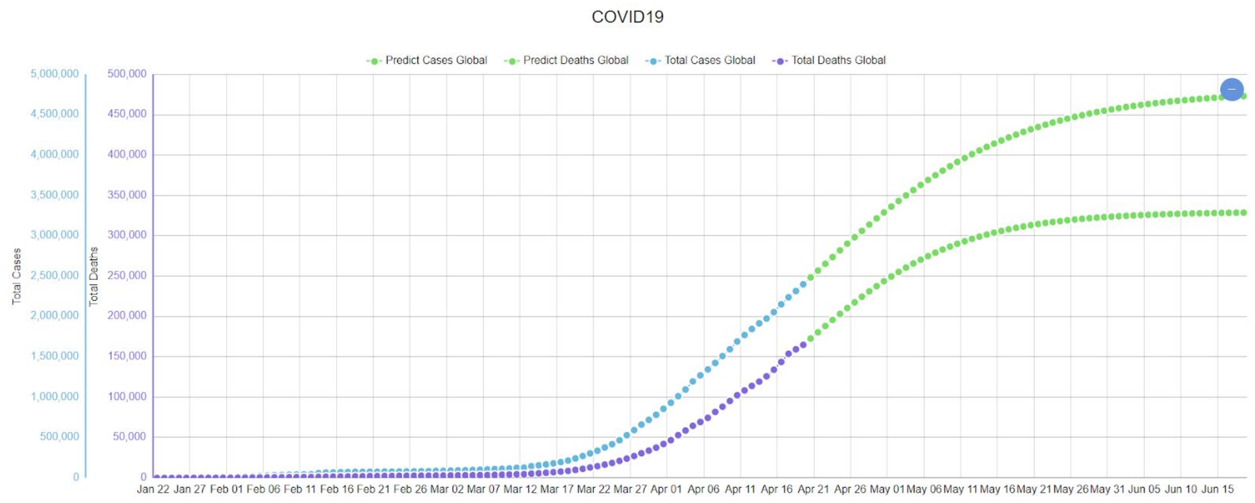


Above: The graph displaying the percentage of google search traffic for 9news, ABC news and 7news compared to Australia's total cases and deaths (Notice that they all peak on Australia's first peak in new cases)

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.

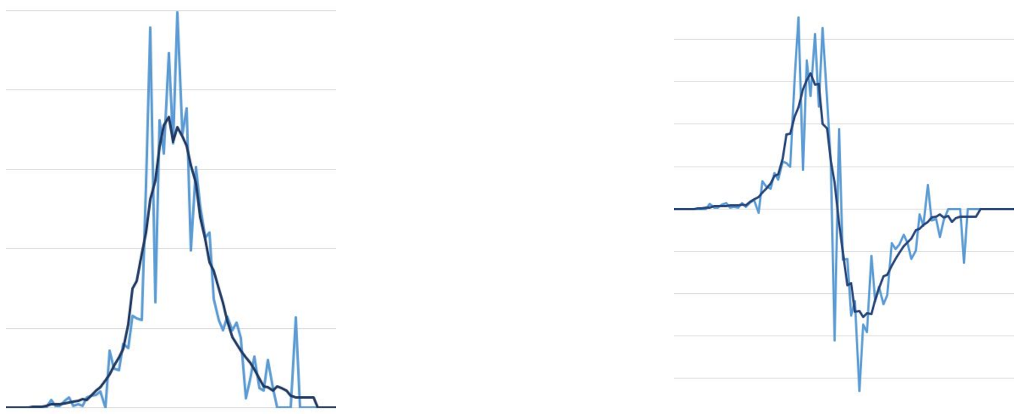
2.2.2 Algorithms

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.

3 Team Organisation

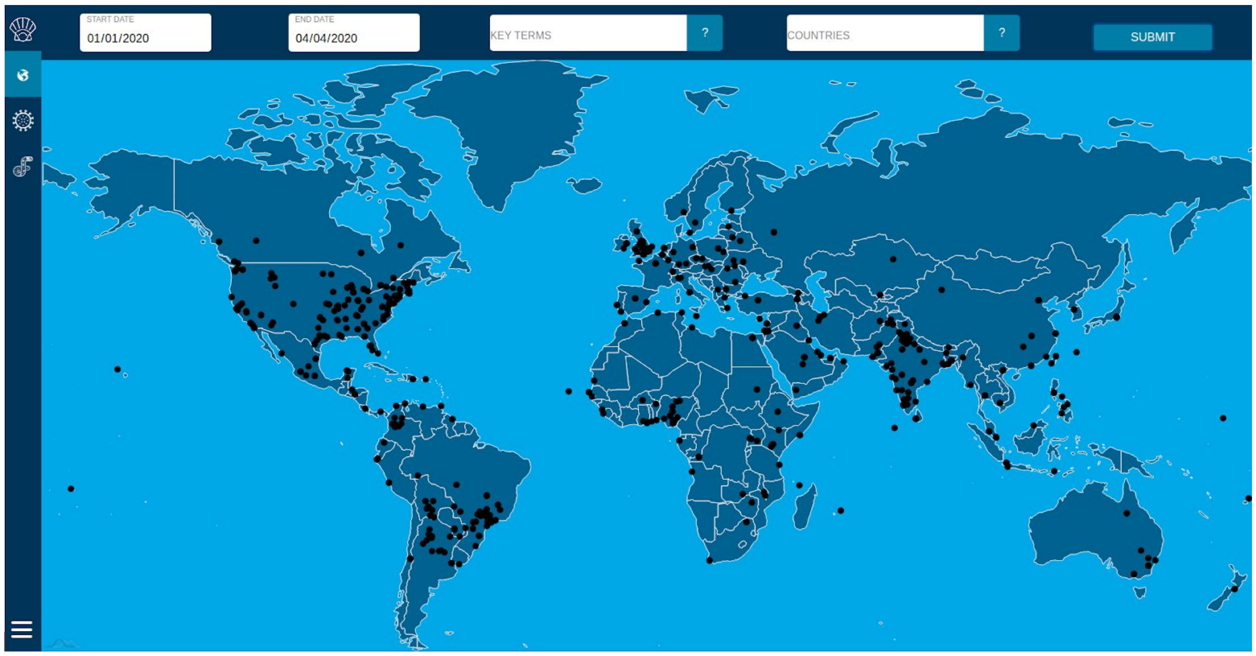
3.1 Responsibilities of the Team

Team Member	Responsibilities
Carlin	Graph section of the frontend, web scraper
Lucas	API, error handling, frontend, ebola backend
Adam	Map section of frontend, web application backend
Marcus	API, orchestrated frontend
Sean	Graph prediction engine, business requirements

3.2 How Did the Project Go?

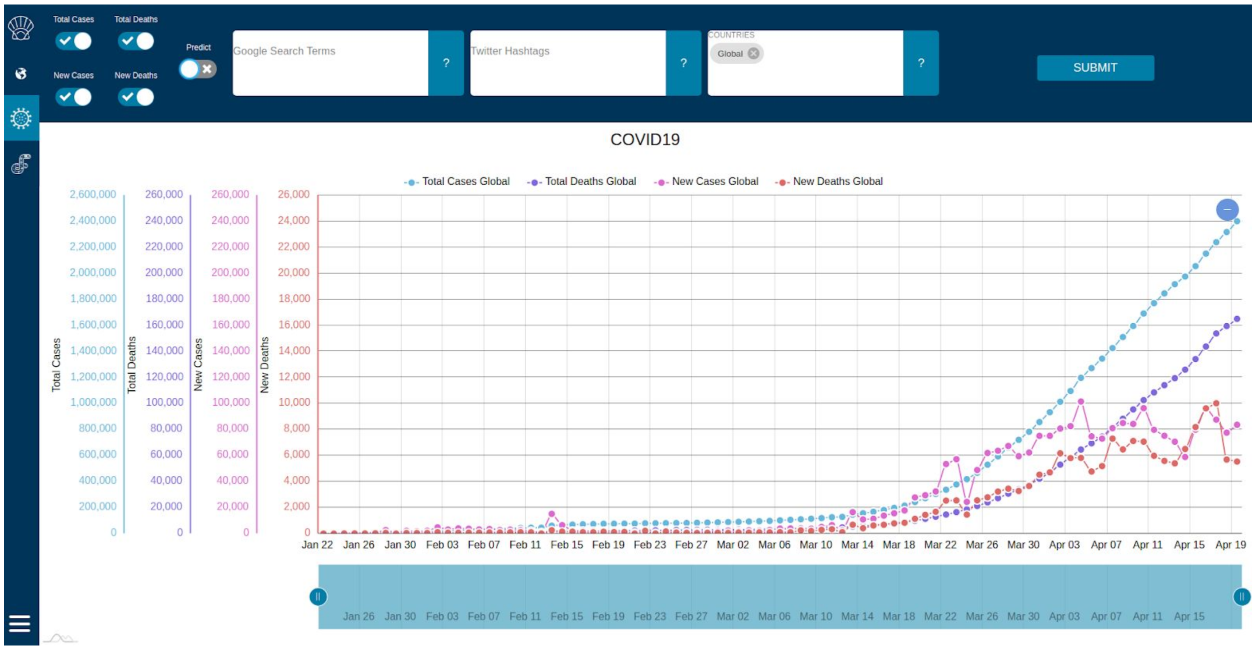
3.2.1 Major Achievements

One of the major achievements of our project was the map that we integrated into the frontend. Our API produced a large amount of data, with our database storing over 30,000 records. So we needed a compact way to view that information that did not overload the user. The map allowed us to display all the data at once, and display information about the data such as its location without cluttering the website or overloading the user with data. This achievement allowed for effective use of our API, and other teams APIs that we integrated into the map.



Above: The map displaying a range of reports from the 1st of January, 2020, to the 4th of April, 2020

Another one of our achievements, and arguably more impressive, was the graphing engine we created. This, much like the map, was able to display a large volume of data easily, but it also was able to facilitate detailed comparisons at a glance. This meant that we were able to continuously integrate more data into the system, and through selective filtering a user could make meaningful conclusions from the data. This, coupled with our prediction engine, helped produce an impressive platform for analysing outbreaks and developing an understanding of the potential future of an outbreak.



Above: The graph showing current data relating to the current COVID-19 pandemic

These two achievements culminate in the final achievement of producing a working prototype of the outbreak analysis platform that we had planned to create at the beginning of this course. It is encouraging to see these plans come into fruition.

3.2.2 Issues/Problems Encountered

Although we managed to deliver a final product that we were happy with, we faced several potential issues that could have threatened this outcome. One major problem was that our API could not give us global data cleanly. At worst, this meant that our app could not interpret whatever API we were using, and at best we would still be left with a map that was less accurate, and less visually appealing, than we were hoping for. Our team managed to write a sanitiser that verifies whatever data we received, allowing our app to interpret it without breaking.Through trial and error, we also managed to find an API which seemed to have the least amount of inaccuracies in global data. Combined with our interpreter, we managed to mostly nullify this issue.

However, there were some goals we did not manage to achieve because of unforeseen circumstances. The most notable of these was implementing pagination for the reports. Originally, we had not anticipated we would need this feature, as nobody in the team had visualised the sheer number of reports that may be returned by any one query. After our initial prototype, we noticed this flaw, and so immediately resolved to implement pagination for the reports. However, after many hours of work we realised that this would be a lot more complicated than originally anticipated, due to the interaction with the map-to-report link. Due to this realisation, we pivoted our work towards more important features, leaving this time-intensive change in our backlog. In future projects, more planning and research would allow us to quickly identify this problem, and plan for it with the design of our app's architecture.

3.2.3 What Kind of Skills Would Have Been Useful?

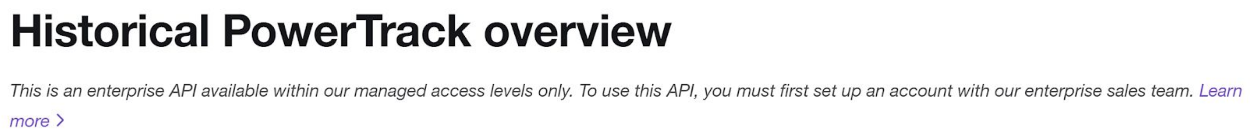
To complete this project, a wide variety of soft, and technical skills are needed, that aren't necessarily provided by the uni. While technical skills allowed us to create a versatile and functional product, the development of our soft skills allowed us to accurately convey the value of what we have actually made to others and communicate well with each other.

None of our team had taken a course in web development, so we had to draw on our previous industry experience so that we could create the app we were aiming for. A few members of our team had a solid understanding of Web App development with React, which enabled our team to both quickly create a quality frontend interface, and provide assistance to the less experienced members of the team to bring them up to speed with any potential problems and bug fixing. Similarly, API development and deployment were skills that nobody in our team had. Part way through the course, we had to research what this skill was, and then plan out the most time efficient and high quality way of implementing this skill. If everyone in our team understood API's, and how to utilise them, before we had started our project, we would have spent significantly less time researching and learning through trial and error.

Another area of skills that would have been useful in the creation of our application would have been some more business focused skills. Whilst most of our degrees have had a focus on technical skills, we found ourselves ill equipped when it came to making business decisions about what our application should do and what audience it should target. More importantly, we would have benefited from some skills relating to presentations and reporting, as we do not currently have the business perspective to know what comes out of these areas.

3.2.4 Things We Would Do Differently

Similar to the Google Trends Data, we were planning to use the Twitter API to retrieve historical Twitter trends. We had researched the Twitter API and knew that the functionality existed for us to achieve our goals, however it required acquiring an Enterprise Twitter Account. This involved filling out an application to be reviewed and approved by Twitter. We submitted an application and were approved but unfortunately, it took too long for Twitter to respond and so we did not have enough time to implement the API into our website. Next time we would understand that some API's require an application process and apply for them much earlier. Fortunately, we were able to implement the Google Trends API functionality so that we could demonstrate our feature as intended.


Historical PowerTrack overview

This is an enterprise API available within our managed access levels only. To use this API, you must first set up an account with our enterprise sales team. Learn more >

Above: Screenshot of the Twitter API feature we intended to use requiring an enterprise account

One front end design feature that we felt would be particularly useful was the addition of autocomplete. When implementing this feature however, it proved difficult to find a convenient tool to use that worked with our Javascript framework and one that did not require copying vast amounts of code. There was one solution that involved importing a Material-UI Autocomplete Component which can be found here: Autocomplete Component. Unfortunately, this component is one of Material-UI's Lab components which means it is still in an experimental stage and has not been finished. What this meant for us, was that it made it more difficult for us to customise what the component looked like in our website. Functionally it worked great and we were happy with the end result, but next time we would have definitely preferred to have conducted more research and perhaps trialled different implementations so that we could find a better one to suit our website.