UNSW
SYDNEY

SENG3011 T1 2021

Design Details

Team: Kool Kats

Mentor: Mirette Saleh

Team members:

Lachlan Stewart

z5207906

Pericles Telemachou

z5214738

Stephen Comino

z3470429

Joanne Chen

z5207491

Weiting Han

z5210581

Table of Contents

API

API Use Cases/Requirements

Use Cases

/disease
A user can use our API /Disease end point to get information on the number of cases and the number of articles (occurrences) on a particular disease in a given time frame. The user is allowed to give a location that they are interested in to get the information pertaining to that location.
A not for profit company could use this information to organise assistance to that country with the accurate data from this api endpoint.
An individual could plan a holiday using this disease data.

/occurrences
A user could use our API /Occurrences endpoint to get the location of a specific disease when they give the disease and timeframe to the api.
They will receive the number of cases of the disease in the location and they will also get the number of articles related to that disease in that location.

If a researcher or business is interested in the number of cases and location of a specific disease they could use this API endpoint to get this information.

/articles
A user could use this API /articles endpoint to get the information of a specific disease in a particular location. This endpoint will return the articles related to that disease from the location given as input.
as any specific location that they are interested in.

/popularDiseases
A user could use the API /popularDiseases endpoint to get a list of the disease with the most cases. The user can choose how many diseases are returned from the api as well

Requirements

Req1 - Return the number of cases for a specific disease given by a user.

Req2 - Return the number of cases and the location for a specific disease given by the user

Req3 - Return the number of articles and cases given a particular disease by the user also let the user specify a location if they want

Req4 - Allow the user to specify a time range for when they want to search for a particular disease

Req5 - Return all the articles related to a particular disease at a particular country in the world

Req6 - Return the location and number of cases of a disease given the disease input by the user.

Req7 - Return a list of the disease with the most amount of cases to the User. Let the user choose how many items they want in this list.

API Design and Implementation

Final API architecture

- Stoplight for designing the REST API and documenting it
- Swagger as a copy of the documentation and for “Try it Out” functionality
- An Amazon EC2 instance (running Ubuntu)
- A Python Flask server running on the EC2 instance to handle incoming API requests.
- A db_functions.py file that contains our functions for retrieving data from the db and formatting it to send back to the user
- Our lambda functions were replaced with a python script that is manually run (at the moment) to update the database
- Amazon’s Cloud database solution for storing any web-scraped data and/or data POSTed by users (RDS MySQL)
- Selenium python library for web scraping
- A RequestData.js ReactJS component containing functions for making requests to the APIs that we used

Choice of Implementation

Stoplight/Swagger

We used Stoplight for construction of the API documentation as it allows multiple users to collaborate and work on it through its GitHub integration as opposed to Swagger which only allows 1 user to edit at a time. Users can create API documentation, mock servers with no specialized knowledge required, and it’s faster than other API tools. We then used Swagger for publishing our API for public use.

Python and Flask

We have chosen Python3 for the development of our server because it is simple to use and there are extensive libraries and modules available for it that are well documented.

Python also has well documented web scraping libraries such as Selenium and Requests as well as parsers such as BeautifulSoup, which is one of the main functions of our server. As a result, it is easier to use and learn as opposed to Node JS, for example.

We used Selenium for web scraping due to its detailed documentation and online resources which made it easy to learn.

Furthermore, most of our group members are also familiar with Python. This meant that we could spend less time learning a new language and more time working on the functionality.

We have chosen to use Python's microframework Flask over Django due to its minimalistic design, simplicity and speed.

EC2 Instance

We have chosen to host our Python server on an AWS EC2 instance as it is easily scalable.

EC2 also runs a virtual operating system which would suit our needs for a server.

Amazon RDS MySQL

We have chosen to use RDS as it was the cheapest option. It allows 750 hours of usage and 20GB of storage at the base tier. RDS was also easy to use and setup.

We used MySQL which is an open source relational database management system. It allows us to store large amounts of text efficiently.

Challenges

One of the challenges we faced was debugging Stoplight. This was due to all of us trying to push with different versions. As there was limited documentation, it took a while for us to figure out the issue.

Another challenge we had was that Stoplight's 'Try it out' function didn't work with our server's static IP address. We worked around this by putting our 'Try it out' functionality onto a Swagger page along with the documentation. In doing this, we were able to make our real API available to users alongside its documentation.

Shortcomings

Shortcomings of this API design include unforeseen costs (AWS costs are based on usage of resources).

As an interpreted script language, Python can be slower compared to other alternatives.

Also, as Flask is single threaded, it can only handle one API request at a time. However, as the API is not expected to be widely used, we think it is sufficient for this project.

API Implementation

Our API documentation can be found on https://app.swaggerhub.com/apis/KoolKats/KoolKats/1.0.

We aimed to make API endpoints that would be functional for the user. We tried to make an endpoint that made effective use of each input parameter.

Endpoints

The endpoints of our API include:

- /articles
- /disease
- /occurrences
- /popularDiseases

All of these endpoints can only accept GET requests with the input as query parameters.

GET /articles

What does it do?

This endpoint gets all articles available scraped from WHO break out news according to search. Results are ordered by date from recent to less recent.

Why is it needed?

This is the core functionality of our API, as mentioned in the requirements.

How it works

Input

Query parameters:

- startDate
- endDate
- Location
- keyTerms

Response


```
{
  "debugInfo": {
    "name": "KoolKats",
    "accessTime": "2021-03-19 04:44:22.416770",
    "serviceTime": "0:00:00.042005",
    "dataSource": "WHO"
  },
  "articles": [
    {
      "date_of_publication": "2009-04-24 00:00:00",
      "headline": "",
      "location": "Mexico",
      "main_text": "The United States Government has reported seven confirmed human cases of Swine Influenza A/H1N1 in the USA (five in California and two in Texas) and nine suspect cases. All seven confirmed cases had mild Influenza-Like Illness (ILI), with only one requiring brief hospitalization. No deaths have been reported.\nThe Government of Mexico has reported three separate events. In the Federal District of Mexico, surveillance began picking up cases of ILI starting 18 March. The number of cases has risen steadily through April and as of 23 April there are now more than 854 cases of pneumonia from the capital. Of those, 59 have died. In San Luis Potosi, in central Mexico, 24 cases of ILI, with three deaths, have been reported. And from Mexicali, near the border with the United States, four cases of ILI, with no deaths, have been reported.\nOf the Mexican cases, 18 have been laboratory confirmed in Canada as Swine Influenza A/H1N1, while 12 of those are genetically identical to the Swine Influenza A/H1N1 viruses from California.\nThe majority of these cases have occurred in otherwise healthy young adults. Influenza normally affects the very young and the very old, but these age groups have not been heavily affected in Mexico.\nBecause there are human cases associated with an animal influenza virus, and because of the
```

geographical spread of multiple community outbreaks, plus the somewhat unusual age groups affected, these events are of high concern.\nThe Swine Influenza A/H1N1 viruses characterized in this outbreak have not been previously detected in pigs or humans. The viruses so far characterized have been sensitive to oseltamivir, but resistant to both amantadine and rimantadine.\nThe World Health Organization has been in constant contact with the health authorities in the United States, Mexico and Canada in order to better understand the risk which these ILI events pose. WHO (and PAHO) is sending missions of experts to Mexico to work with health authorities there. It is helping its Member States to increase field epidemiology activities, laboratory diagnosis and clinical management. Moreover, WHO's partners in the Global Alert and Response Network have been alerted and are ready to assist as requested by the Member States.\nWHO acknowledges the United States and Mexico for their proactive reporting and their collaboration with WHO and will continue to work with Member States to further characterize the outbreak.",
      "termFound": "influenza",
      "url": "https://www.who.int/csr/don/2009_04_24/en/"
    }
]
}

GET /disease

What does it do?

This endpoint gets the number of cases of a disease within the date range. It also provides the number of articles that the specified diseases were found in as well as the total number of articles. Results are ordered by the number of cases for each disease.

Why is it needed?

If the user wants to get specific case related information i.e. number of cases in a location or time period, they can make this request to do that. Users can also get the proportion of articles that reference the specified diseases.

How it works

Input

Query parameters:

- keyTerms
- startDate
- endDate
- Location (optional)

Response

```
{
disease: [{
        name: disease,
        cases: int,
        occurrences: int
        }],
totalArticles: int,
debugInfo: {
        name: 'KoolKats',
        accessedTime: string,
        serviceTime: string,
        dataSource: 'WHO'
        }
}
```

GET /popularDiseases

What does it do?

Gets the most popular diseases within the date range provided. The number of diseases returned is determined by the user. Results are ordered by the number of articles that each disease is found in.

Why is it needed?

For users that want a breakdown of popular disease occurrences in date range.

How it works

Input

Query parameters:

- startDate
- endDate
- Location (optional)
- numDiseases (optional)

Response


```
{
  "rankings": [
    {
      "name": "Ebola virus disease",
      "occurrences": 2,
      "cases": 154
    },
    {
      "name": "Cholera",
      "occurrences": 1,
      "cases": 65
    },
    {
      "name": "Rift Valley fever",
      "occurrences": 1,
      "cases": 30
    }
  ],
  "debugInfo": {
    "name": "KoolKats",
    "accessTime": "2021-03-19 04:44:22.416770",
    "serviceTime": "0:00:00.042005",
    "dataSource": "WHO"
  }
}
```

GET /occurrences

What does it do?

This endpoint gets the occurrences of diseases, separated by countries (heatmap of occurrences). Results are ordered by the number of articles that each disease is found in.

Why is it needed?

It shows the distribution of the disease and allows users to easily determine in which countries diseases are occurring the most.

How it works

Input

Query parameters:

- keyTerms
- startDate (optional)
- endDate (optional)

Response


```
{
  "locations": [
    {
      "disease": "Yellow Fever",
      "name": "Democratic Republic of Congo",
      "occurrences": 2,
      "cases": 20
    },
    {
      "disease": "Yellow Fever",
      "name": "Islamic Republic of Pakistan",
      "occurrences": 1,
      "cases": 30
    },
    {
      "disease": "Yellow Fever",
      "name": "Haiti",
      "occurrences": 1,
      "cases": 14
    }],
  "debugInfo": {
    "name": "KoolKats",
    "accessTime": "2021-03-19 04:44:22.416770",
    "serviceTime": "0:00:00.042005",
    "dataSource": "WHO"
  }
}
```

Web Application (DiSeeker)

Web Application Use Cases

Target Audience:

- Not-for-profits e.g. Doctors Without Borders, Red Cross, Médecins Du Monde
- Government health departments to track the disease response of other countries

Other possible uses:

- Individuals who are travelling interstate within Australia and need to identify travel restrictions
- Individuals looking to travel to different countries to see disease outbreaks and impact of COVID-19

Persona: Sam (Strategic Planner from Doctors without borders)

Goal: To determine where to allocate the organisation's resources to have the most impact for the delivery of vaccinations

- User Story 1: As a strategic planner I would like to be able to specify multiple different diseases so that I can see how to allocate resources in the world for different diseases
    Example
    - Sam also wants to find out about the countries that have been affected by Yellow Fever to plan his NFPs funding
    - He goes back to the "Disease", "Start Date" and "End Date" filters
    - He adds the disease Yellow Fever to the disease list and broadens the date range to the last year
    - The map automatically updates to reflect his changes
    - Sam can then see which countries have been worst affected by Yellow Fever and factor it into his NFPs budget planning

- User Story 2: As a strategic planner I would like to be able to see which areas in the world are worst affected by a given disease so that I can plan the budget of the NFP I work for effectively
    Example
    - Sam wants to find out which countries have been most affected by COVID in the last year, so that he can plan how to allocate his NFPs funding.
    - He goes onto DiSeeker and narrows the disease to "COVID-19" and restricts the date range to the last year
    - After the map updates, he clicks on the region that is the most heavily shaded to bring up more detailed information about it
        - Cases
        - Vaccination percentage (COVID)
        - Average change in cases by month

- User Story 3: Planning fundraising events in Australia and has to travel interstate.

I would like to be able to see each state's travel restrictions so that I can plan my travel ahead of time

Example

- Sam lives in Australia and will also be travelling interstate for his work, so he wants to find out what the travel restriction are between states
- He goes onto DiSeeker
- Clicks on Australia
- He clicks on the "Restrictions" button and select "VIC" from the drop down
- The current restrictions for that state come up
- He can then plan his travel around the restrictions

Web Application Design and Implementation

Final Web Application Architecture

- React JS framework for the main frontend development
- React-bootstrap for styling
- Yarn as the dependency manager for the web application
- AWS EC2 Instance for hosting the web application

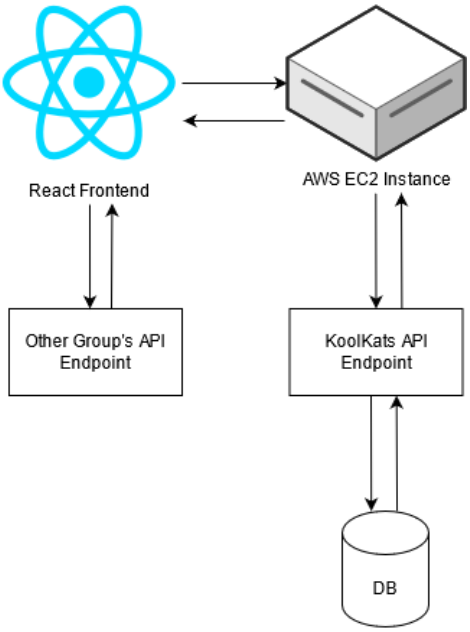


Figure 1 - Web Application Architecture

Choice of Implementation

AWS EC2

We have decided to use an AWS EC2 instance to host our web application because it is easily integratable into the instance we have to host our API endpoints.

React JS

React JS allows us to create reusable components, allowing more robust applications. Also, React JS has a virtual DOM which allows the UI to update more quickly and efficiently. Furthermore, all of our team members have used React JS before and are more familiar with it. As a result, there would be less of a learning curve allowing more features to be implemented.

React Modules

React simple maps

React simple maps provides pre-made components for us to easily create a zoomable map that we can modify using the data from our API

React high charts

React high charts provide a template component for us to easily insert information and create graphs. By using this, we are able to dynamically update our charts (pie chart, line graph) as required.

Bootstrap

Bootstrap is a frontend framework with templates for UI elements such as buttons and forms. Bootstrap is open source and free, allowing us to easily design the user interface.

Yarn

We have decided to use Yarn as our dependency manager of npm due to its speed and efficiency. Yarn also allows us to autoclean to clear dependencies that aren't necessary, saving storage in the node_modules.

APIs

We decided to use 4 APIs, including the one we created, 2 from SENG3011 groups and 1 external API.

KoolKats API (SENG3011 Group)

We decided to use KoolKats API because we know the API well and have designed some of its endpoints to suit our needs. Our API also aggregates data such as case numbers to provide more useful insights.

Sour Dough's API (SENG3011 Group)

We decided to use Sour Dough's API to provide COVID vaccination percentage data for each country. We feel that this API will provide further valuable information to our users that'll enable them to not only recognise where diseases are most prevalent but also which countries have or haven't effectively rolled out COVID vaccinations.

Brickwalls API (SENG3011 Group)

We decided to use Brickwalls API to provide data for Australia interstate restrictions.

Disease.sh API

We decided to use disease.sh API as it provides up to date information on COVID-19 data on the world and different countries. Disease.sh uses data sources from John Hopkins University.

Shortcomings

As a lot of the API's used are external, we have little control of how updated their data is as well as the consistency of the data formats.

Challenges

One of the main challenges of developing DiSeeker was the development of all the visual graphs and charts as none of us had experience dealing with charts in the frontend before.

Using the APIs of other groups was also a challenge that we faced. Both groups had the CORs policy issue meaning we could not use their API. Only one of the two groups ended up fixing the issue so for the other group we had to use static data structured from their example response.

Another challenge we had was managing information received from external data sources as it was difficult to obtain valid data form the various APIs used.

We were also alerted that the WHO website that we were scraping would no longer be accessible from April 2021. As a result we had to speed up the process of web scraping so that we had the latest articles in our database before the web page was taken down.

Key Achievements/Benefits of Design

- Achievement: Scraping all articles from both the old WHO website and the new one
- Achievement: Incorporating 4 APIs into our design in total including our API, two other SENG3011 groups' APIs and a 3rd party API
- Benefit: Gives the user a visual representation of disease spread
- Benefit: Gives the user the ability to adjust the output with the "Disease" and "Time Period" parameters
-

Implementation

The application was built using React and Bootstrap due to their simplicity. The React Multiselect Dropdown package was used for the disease input field and the React Date Picker package was used for the end and start date selector. The React HighCharts

library was used for all graphs in the information panel such as the pie graph. React Simple Maps was used for the heat map.

Some example (figures 2-7) uses of DiSeeker:

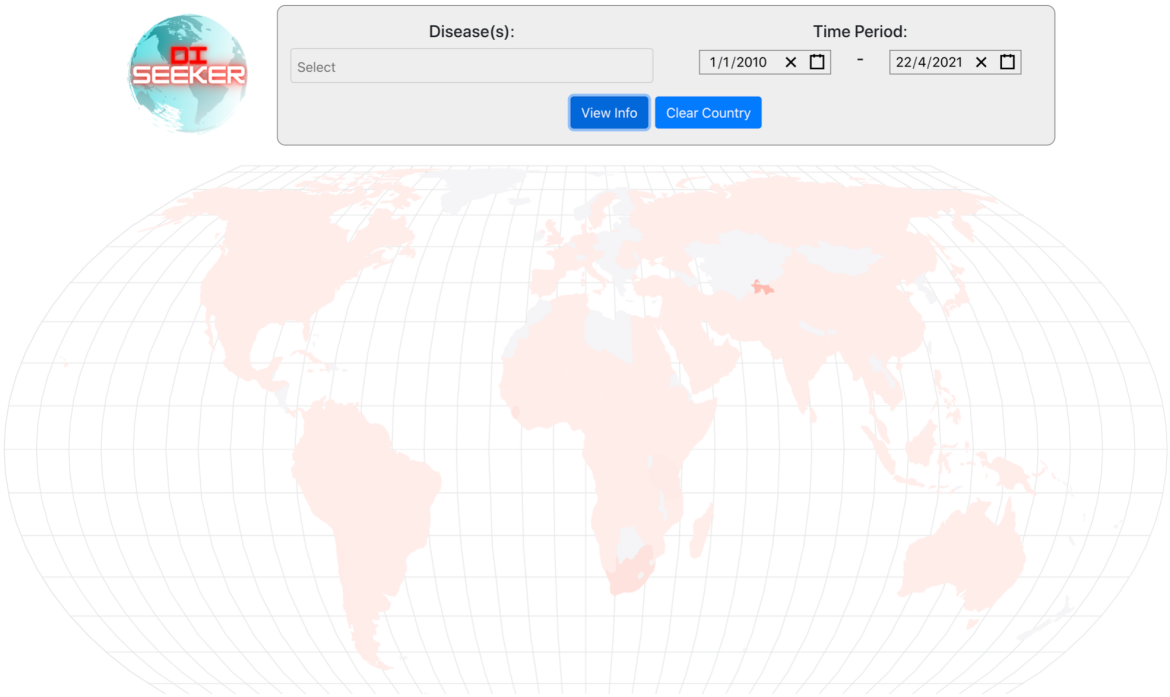


Figure 2 - DiSeeker home page

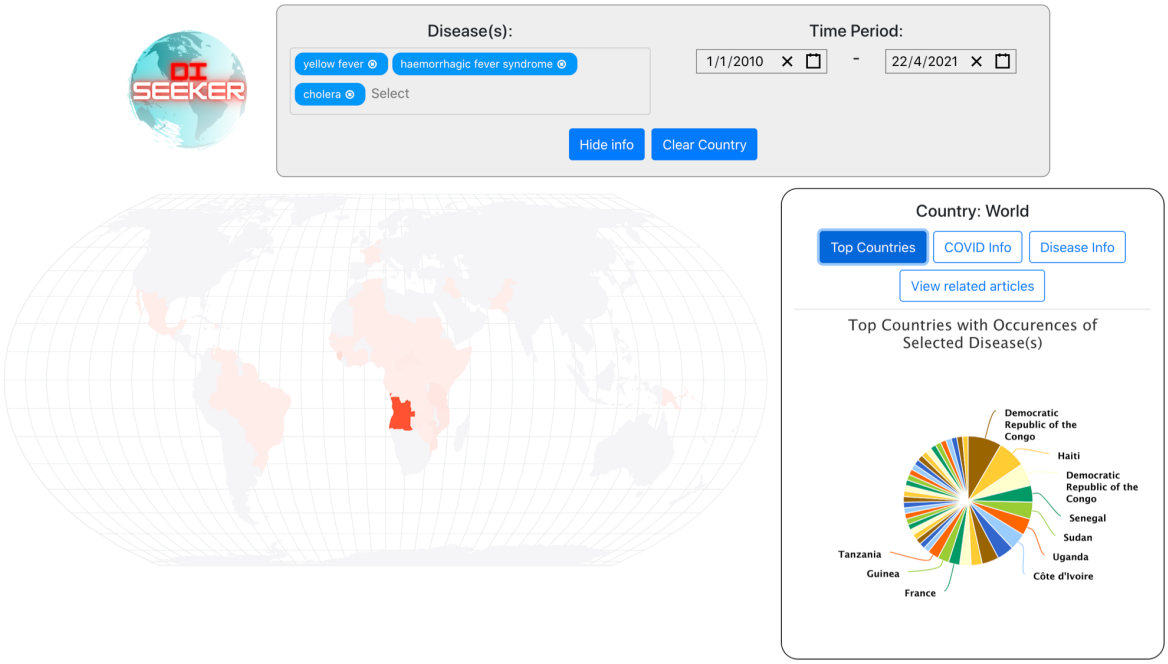


Figure 3 - DiSeeker diseases specified and information panel

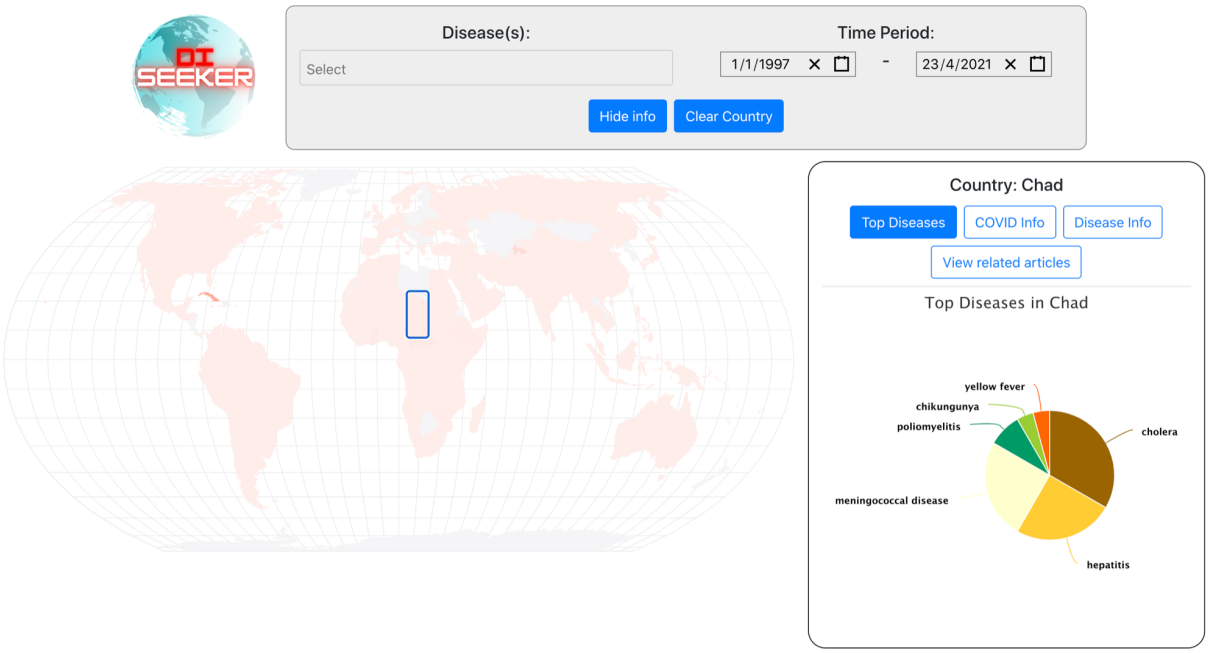


Figure 4 - DiSeeker information panel with country selected

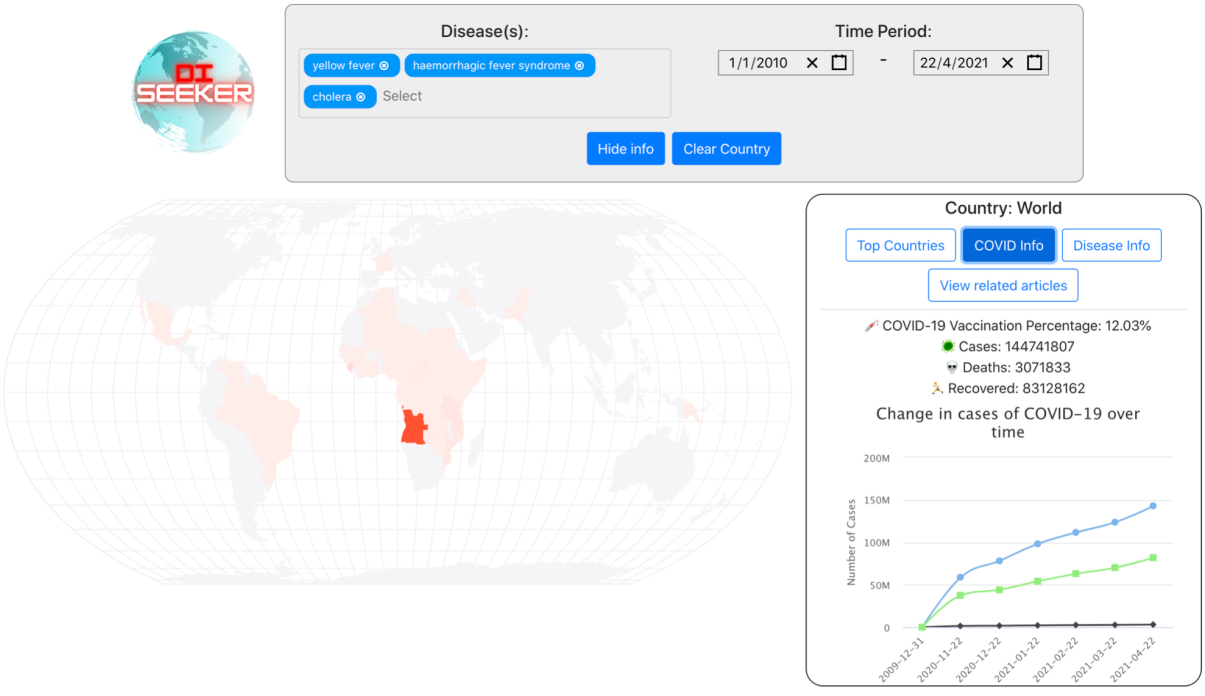


Figure 5 - DiSeeker COVID information panel

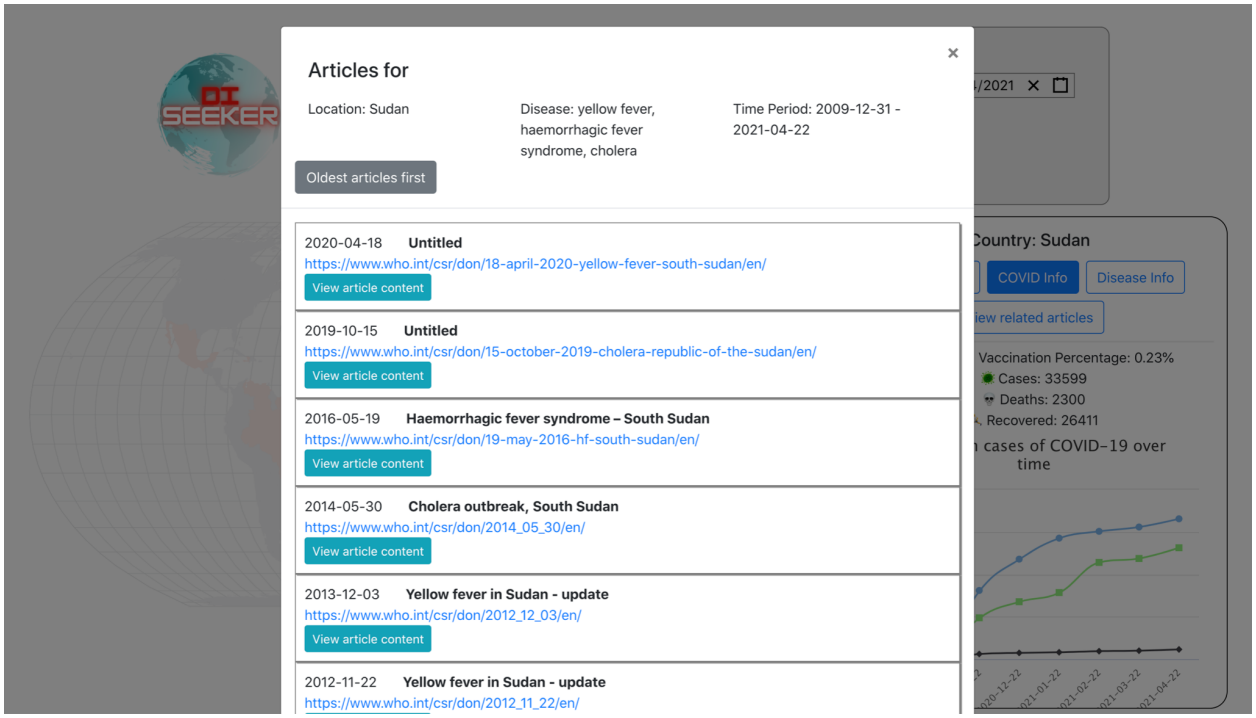


Figure 6 - DiSeeker articles popup

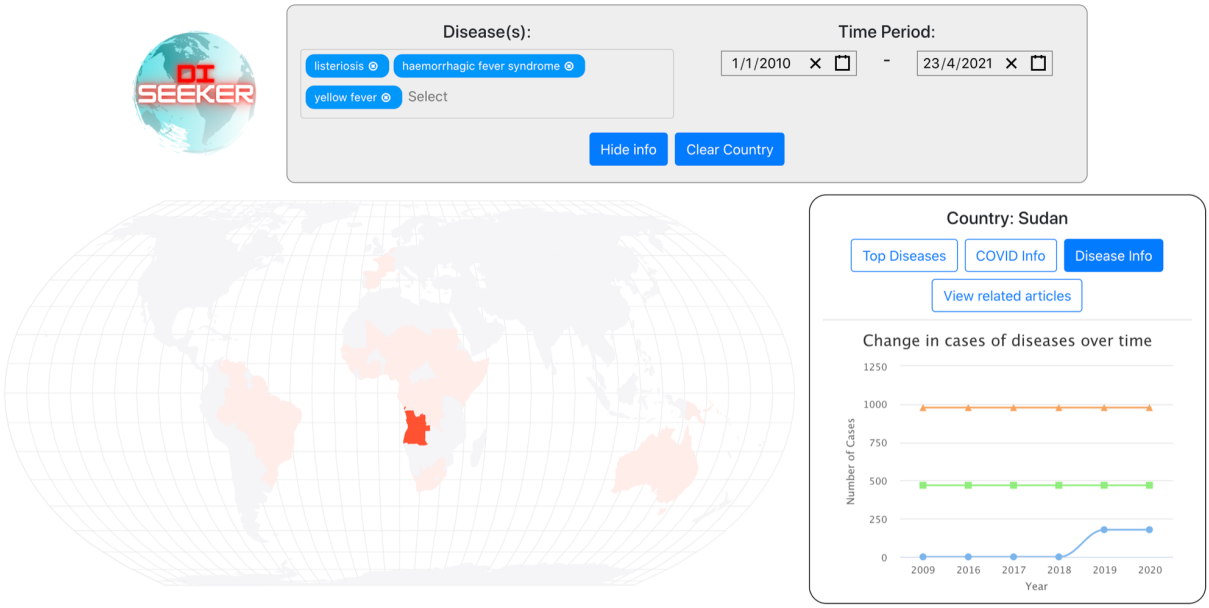


Figure 7 - DiSeeker information panel with case changes over time

Project Management

Project Plan

We plan on using Microsoft Teams for communication and meetings as well as Trello for project management as seen in Figure 1. We have chosen these tools as members within the group are familiar with them and have found them to be easy to use and effective in the past. We have assigned team member responsibilities as seen in the table below. We have assigned one or two members to be responsible for each component of the project, however we feel that all members can contribute to each component. We have used the skills and personal preferences of each team member when assigning responsibilities. When there is no SENG3011 lecture allocated, we have allocated that time for team meetings as most team members are free during this time. We also plan to meet before/after the tutorial meeting depending on team member availability.

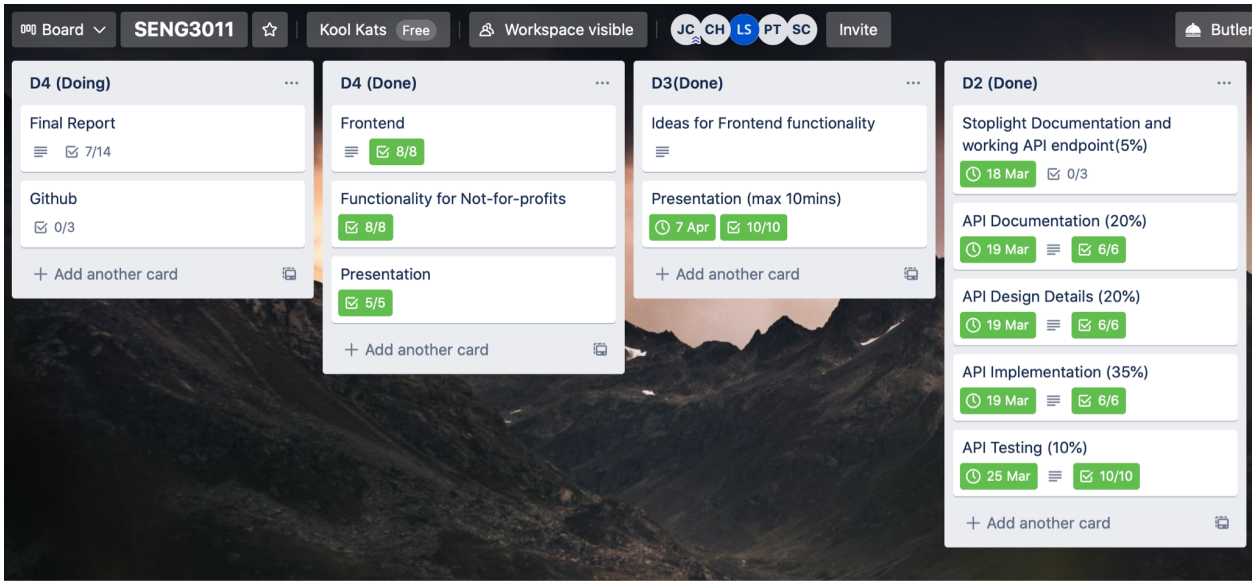


Figure 1 - Group Trello Board

Team member responsibilities

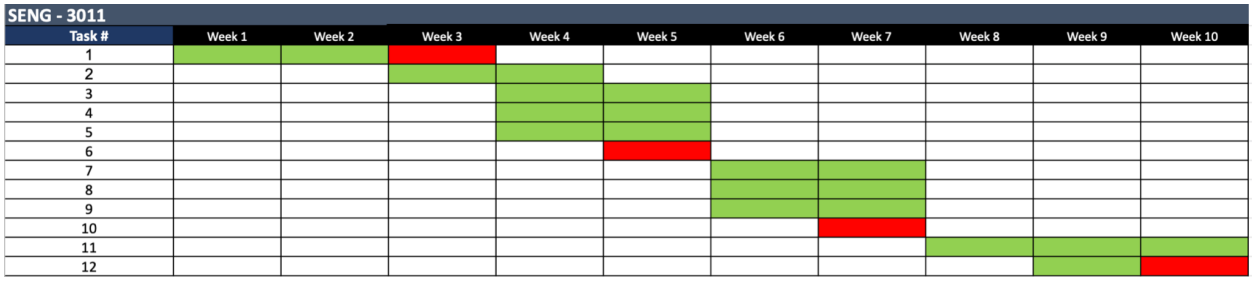
	Report	API Docs	API Testing	API Development	DB Management	Web Scraping	Frontend
Peri T	✓		✓	✓	✓		✓
Joanne C	✓	✓	✓				✓
Lachlan S	✓		✓	✓		✓	✓
Catherine H	✓		✓				✓
Stephen C	✓		✓	✓		✓	✓

Project schedule

Project deadlines

Week 3 - D1 (Initial api design, project management plan)
Week 4 - Initial architecture setup, Web scraping, api documentation, api implementation
Week 5 - D2 (API Documentation, API development)
Week 6 - Frontend design, integrate other group's APIs
Week 7 - Frontend implementation
Week 8 - D3 (Project prototype demo)
Week 9 - More frontend, report
Week 10 - D4 (Final demo, final report)

Gantt Chart



Task #	Task
1	D1 (Initial api design, project management plan)
2	Initial architecture setup
3	Web scraping
4	API documentation
5	API implementation
6	D2 (API Documentation, API development)
7	Frontend design
8	Integrate other group's APIs
9	Frontend implementation
10	D3 (Project prototype demo)
11	Improve Frontend
12	D4 (Final Report)

Key	
Green	In Progress
Red	Due Date

Testing Documentation

Overview of Testing

In order to ensure our API was working correctly and as expected, there were a few main things to check:

- API Accuracy
    - Our group decided to test our accuracy through manual checking of WHO articles and comparing them to our API output.
- Database accuracy
    - We tested the accuracy of the database through manual checking of WHO articles and what was in the database
- Speed
    - Our group decided to test the speed of our API through the use of the log file/debug output that comes with the API response.

Testing Environment

All tests were done with both a local server and the API server we deployed on AWS. All data and articles used for testing came from the WHO website.

Limitations of Testing

Because of the complexity of the data we had to manually check and test our data. We could only test a subset of the data and we assumed correctness with the rest as they were all collected in the same way.

For performance testing we could only test factors that we had direct control over (i.e. time taken for our functions to process requests, time taken to fetch data from database, etc). We couldn't factor in other bottlenecks in the process like time taken to reach the AWS server, application performance (i.e. Postman), etc.

Performance Testing - Manual Checking for API Accuracy

One of the testing processes we used was manual checking for output correctness. This involved going through articles and comparing our results with API output. This would check the accuracy of our API in all endpoints.

We first entered input into our api and checked the resulting output Headlines and article text with the WHO website headlines and article text to see if they are identical.
Because of the large amount of data returned, only a subset of output from the command was tested. This test assumes that all other data will return the same result because the data for other articles are received and processed the same way by the API.

GET /articles

Input (Anthrax United States of America From: 2000-01-10T00:00:00 To:2020-01-10T00:00:00)	API Output From Input	WHO Title:	Comparison of Text	Accuracy
	2001 - Anthrax in the United States (State of Florida)	2001 - Anthrax in the United States (State of Florida)	Exactly the same	100%
	2001 - Anthrax in the United States (State of Florida) - Update	2001 - Anthrax in the United States (State of Florida) - Update	Exactly the same except inline url tag	100%
	2001 - Anthrax in the United States - Update 2	2001 - Anthrax in the United States - Update 2	Exactly the same	100%
	2001 - Anthrax in the United States - Update 3	2001 - Anthrax in the United States - Update 3	Exactly the same	100%
	2001 - Anthrax in the United States - Update 4	2001 - Anthrax in the United States - Update 4	Exactly the same	100%
	2001 - Anthrax in the United States - Update 15	2001 - Anthrax in the United States - Update 15	Exactly the same	100%

The above output is only a subset of the results (too much to put into the table).
Total Accuracy: 100%
Overall, getting the articles and their data (main text, headline, date of publication etc.) through our API has been fairly accurate. It is also accurately mapped to the right locations and diseases. The number of articles found for this case was also accurate (17 articles found).

GET /disease

Input	Manual Checking	API Output	Accuracy
Yellow Fever France From: 2015-01-01T00:00:00 To: 2021-01-01T00:00:00	Cases: 6 Articles: 3	Cases: 19 Articles: 3	Cases: +216% Articles: 100%
Cholera Mexico From: 2013-01-01T00:00:00 To: 2021-01-01T00:00:00	Cases: 184 Articles: 4	Cases: 157 Articles: 1	Cases: -17.2% Articles: -300%
Anthrax From: 2000-01-01T00:00:00 To: 2021-01-01T00:00:00	Cases: 23 Articles: 17	Cases: 45 Articles: 17	Cases: +95.7% Articles: 100%
Dengue Fever From: 2000-01-01T00:00:00 To: 2021-01-01T00:00:00	Cases: 686742 Articles: 37	Cases: 147800 Articles: 21	Cases: -364.6% Articles: -76.2%
Japanese encephalitis From: 2000-01-01T00:00:00 To: 2021-01-01T00:00:00	Cases: 1145 Articles: 1	Cases: 1235 Articles: 1	Cases: +7.9% Articles: 100%

Average Case Accuracy: -12.44%
Average Article Accuracy: -15.24%

Overall, the number of cases and articles have been relatively accurate (with some outliers). However, generally, our API would undercount the articles and overcount cases. Cases have a higher chance of being overcounted as our API finds the number of cases within an article and aggregates it. However, some articles consist of the number of cases ‘up to date’, causing some errors. Articles on the other hand have a higher chance of being under counted as we match exactly to the disease name and location.

GET /occurrences

Input	Manual Checking	API Output	Accuracy
keyTerms: chikungunya startDate: 2018-01-01T00:00:00 endDate: 2018-12-31T23:59:59	Mombasa, Kenya (1) Sudan (1)	Kenya, Mombasa (1) Sudan (1)	100%
keyTerms: yellow fever startDate: 2017-01-01T00:00:00 endDate: 2019-12-31T23:59:59	Mali (1) Nigeria (5) Bolivarian Republic of Venezuela (1) Brazil (12) Kingdom of the Netherlands (1) Republic of the Congo(1) France (2) Suriname (1)	Brazil (3) France (2) Republic of the Congo (1) Mali (1) Netherlands (1) Nigeria (1) Suriname (1) Venezuela (1)	For “Brazil” and “Nigeria”, there are more than one articles each year, but only count once per year. 6/8 = 75%
keyTerms: Rabies startDate: 2000-01-01T00:00:00 endDate: 2020-12-31T23:59:59	France (1)	France (1)	100%
keyTerms: Hepatitis startDate: 2017-01-01T00:00:00 endDate: 2018-12-31T23:59:59	Namibia (1) Nigeria (1) The Americas and Europe (1) Niger (1) Chad (1)	Chad (1) Namibia (1) Niger (1) The Americas and Europe (1)	4/5 = 80%

Total Accuracy: 88.75%

Overall, there are some inconsistencies between the actual numbers and the number our API returns. However, the accuracy of our endpoint is quite high and would work for rough estimates.

GET /popularDiseases

Input	Manual Checking	API Output	Accuracy
2017-01-01- to 2019-12-31 (France)	Yellow fever: 2017 (1), 2018 (1) Dengue fever: 2018 (1), 2019 (1) Chikungunya: 2017 (1) Rift Valley fever: 2019 (1) Zika virus: 2019 (1)	Dengue fever (2) Yellow fever (2) Chikungunya (1) Rift Valley fever (1)	Zika virus not found in API 6/7 = 86%
2014-01-01 to 2016-12-31 (Spain)	Ebola Virus disease: 2014 (1) Chikungunya: 2015 (2)	Chikungunya (2) Ebola virus disease (1)	3/3 = 100%
2001-01-01 to 2004-12-31 (Nigeria)	2004: Cholera (1), Meningococcal disease (1) 2001: Cholera (2)	Cholera (3) Meningococcal disease (1)	4/4 = 100%
2010-01-01 to 2010-12-31	2010: Influenza (33), Cholera (7), Yellow fever (6), Rift Valley fever (3), Crimean-congo Haemorrhagic fever (1), Meningococcal disease (1), Plague (1)	Influenza (30) Cholera (7) Yellow fever (5) Rift Valley fever (3) Crimean-congo Haemorrhagic fever (1) Meningococcal disease (1) Plague (1)	44/48 = 92%
2002-01-01 to 2002-12-31	2002: Meningococcal (14), Cholera (14), Acute respiratory syndrome (7), Yellow fever (8), Dengue haemorrhagic fever (7), Haemorrhagic fever syndrome (5), Leishmaniasis (4), Plague (2), Acute neurological syndrome (1)	Meningococcal disease (12) Cholera (11) Acute respiratory syndrome (7) Yellow fever (7) Dengue haemorrhagic fever (6) Influenza (6) Haemorrhagic fever syndrome (5) Leishmaniasis (4) Plague (2) Acute neurological syndrome (1)	54/61 = 89%

Total Accuracy: 93.4%

Overall, the accuracy of this endpoint is relatively high.

Manual Checking for Database Accuracy

One of the testing processes we used was manual checking for database and web scraping correctness. This involved going through a few articles and comparing the data, headline and text with the actual WHO article.

Manual Checking Article	API response - Database	Accuracy
Headline: 2001 - Cholera in South Africa Date of Publication: 16 March 2001 Main text: Find at Webpage Location: South Africa Disease: Cholera	Headline: 2001 - Cholera in South Africa Date of Publication: 16 March 2001 Main text: Find at Webpage Location: South Africa Disease: Cholera	100%
Headline: Avian influenza - situation in Egypt - update 39 Date of Publication: *29 December 2010* Main text: Find at Webpage Location: Egypt Disease: Avian Influenza	Headline: Avian influenza - situation in Egypt - update 39 Date of Publication: *29 December 2010* Main text: Find at Webpage Location: Egypt Disease: Avian Influenza	100%
Headline: Cholera in Haiti - update 3 Date of Publication: *17 November 2010* Main text: Find at Webpage Location: Haiti Disease: Cholera	Headline: Cholera in Haiti - update 3 Date of Publication: *17 November 2010* Main text: Find at Webpage Location: Haiti Disease: Cholera	100% - Does not include the attached pdf on the database
Headline: Pandemic (H1N1) 2009 - update 108 Date of Publication: *9 July 2010* Main text: Find at Webpage Location: No Specific location Disease: No Specific location	Headline: Pandemic (H1N1) 2009 - update 108 Date of Publication: *9 July 2010* Main text: Find at Webpage Location: No Specific location Disease: No Specific location	100% - Does not include links and formatted table data. Only TEXT

Total Accuracy: 100%

Overall, all the necessary text is loaded onto the database. However, table data, image data and formatted data do not get loaded onto the database.

Non-Functional Testing - Speed of API Response

One of the testing processes we used was manually checking the speed of our API response. This was done by recording the 'service time' property in the log file/debug output that comes with the API response, this measures the time difference between when the request is received at the server and when the response is sent out to the client. By doing this, we were able to measure the average, minimum and maximum time it took for our different request types to be processed at the server.

For each request we tried to maximise the size of the response to simulate a worst case scenario (i.e. broadest date range, not limiting disease names where possible, specifying countries with lots of database entries, etc)

Request Tested	Minimum	Average (of 5)	Maximum
/disease (disease?startDate=1970-01-01&endDate=2021-01-01&location=United States)	22ms (first test)	56ms (first test with 182ms outlier)	182 ms (first test seems like an outlier)
	24ms (second test, 10 requests)	23ms (second test, 10 requests)	24ms (second test, 10 requests)
/articles (articles?startDate=1970-01-01&endDate=2021-01-01&location=United States)	6.2 ms	6.5 ms	7.1 ms
/occurrences (occurrences?keyTerms=hepatitis, cholera, influenza)	5.9 ms	6.0 ms	6.2 ms
/popularDiseases (popularDiseases?startDate=1970-01-01&endDate=2022-01-01&numDiseases=100)	8.9 ms	9.2 ms	9.7 ms

Overall, our responses generally stayed under 30ms. We observed that postman took as little as 300ms and sometimes over 1s to complete a request so there are obviously other bottlenecks outside of our control that extend the time taken to receive a response. A 30ms response will likely be imperceptible to the user and not trigger any timeout errors, therefore we think it's a reasonable amount of time to process an API request at the server.

We observed outliers in the first test for the "disease" endpoint, however, this might be a result of establishing the initial connection or some other rare factor at the server. Subsequent testing produced more consistent results that were more in line with what we were expecting (although they still took at least twice as long as the highest service times from the other endpoints, so there could be further performance optimisations to be made for the "disease" endpoint).

Issues Found Through Testing and Improvements Made

Through testing, we found errors in our API endpoints and noticed some inconsistencies between our API data and actual data. The main issue we found was that we only added articles to our database if they had a unique article name, which was not true as WHO does contain articles that had duplicate names with differing article content. As we found this close to the deadline, it was too late to scrape and put into our database.

Another issue we found was that not all articles were added to the database. This was due to human error as we had duplicate data and accidentally deleted the database when we tried to remove the duplicates. When we recreated the database from a snapshot the data that we thought we had already added was no longer there. As a result, we did a second round of scraping to get more data for our database.

Another issue we found was that our system for extracting the country name from the data was flawed. To improve this, we went through our data semi-manually in order to correct all the instances of incorrect “Country” values.

Conclusion

Overall, we did testing for database accuracy, API accuracy and speed. Through testing, we were able to find issues and inconsistencies and hence improve our API.

Project Reflection

Major Achievements

- Incorporating 4 APIs into out application including our own, other SENG teams and
- Dynamic graphs/charts in our web application
- Good feedback on improvement from the 2 demonstrations/deliverables

Issues/Challenges encountered

- Talking to other groups about their APIs
- Encountering bugs with our frontend and backend
- Group communication
- Finding a good time to meet for each team member was hard initially.
- Setting up AWS to host website was challenging

Skills we wished we had

- Better at Web scraping
- Knowing how to set up an API server. Setting up the AWS server was difficult, we found it hard to do in AWS. We had to familiarise ourselves with tools that we did not know about, like uWSGI and nGINX
- We started this project with no experience creating API endpoints so we feel it would have been better if we had some more experience.

Looking back

Looking back we think we should have organised better at the beginning. This would have saved us lots of time later on in the term when we had lots to do.

We could have also started ideating our web application earlier. That way we would've had more time to work on our product and could've got more functionalities working.

Overall

Overall, the project went great and we achieved a lot. Most members had experience using APIs and found it interesting to be on the other side by creating the API. We also enjoyed the process of learning how to integrate different APIs into our application.