



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	4
API Use Cases/Requirements	4
Use Cases	4
Requirements	4
API Design and Implementation	5
Final API architecture	5
Choice of Implementation	5
Stoplight/Swagger	5
Python and Flask	6
EC2 Instance	6
Amazon RDS MySQL	6
Challenges	6
Shortcomings	7
API Implementation	7
Endpoints	7
GET /articles	8
GET /disease	10
GET /popularDiseases	11
GET /occurrences	12
Web Application (DiSeeker)	13
Web Application Use Cases	13
Web Application Design and Implementation	14
Final Web Application Architecture	14
Choice of Implementation	14
AWS EC2	14
React JS	15
React Modules	15
React simple maps	15
React high charts	15
Bootstrap	15
Yarn	15
APIs	15
KoolKats API (SENG3011 Group)	15
Sour Dough's API (SENG3011 Group)	15
Brickwalls API (SENG3011 Group)	16
Disease.sh API	16

Shortcomings	16
Challenges	16
Key Achievements/Benefits of Design	16
Implementation	16
Changes	20
Deliverable 2	20
Deliverable 4	20

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.\n\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.\n\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.\n\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.\n\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.\n\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.\n\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.\n\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

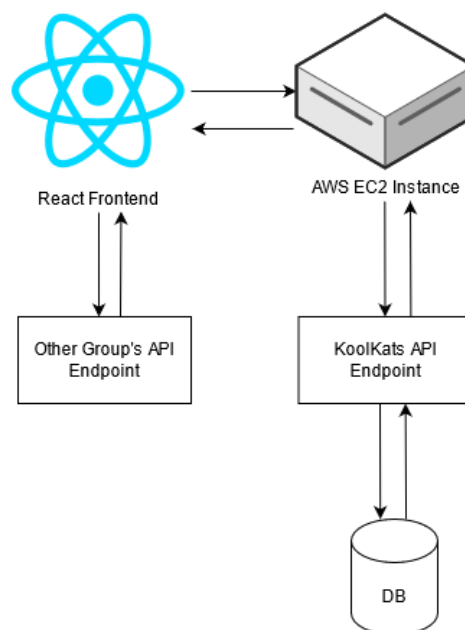


Figure 1 - Web Application Architecture

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 from 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:

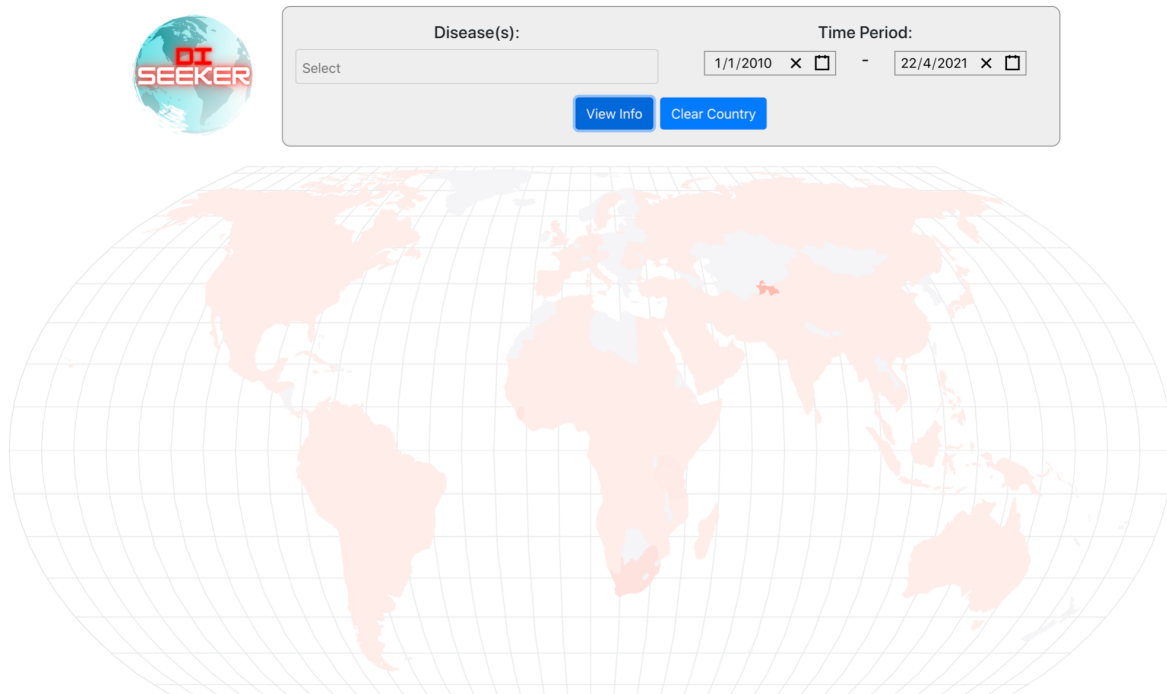


Figure 2 - DiSeeker home page

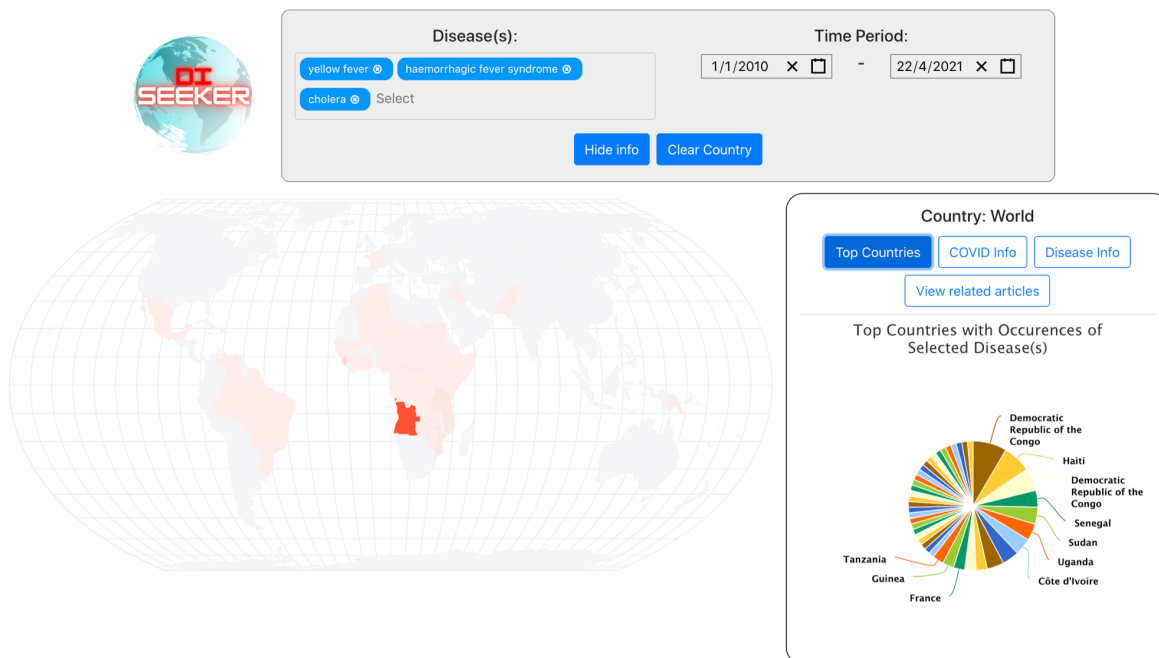


Figure 3 - DiSeeker diseases specified and information panel

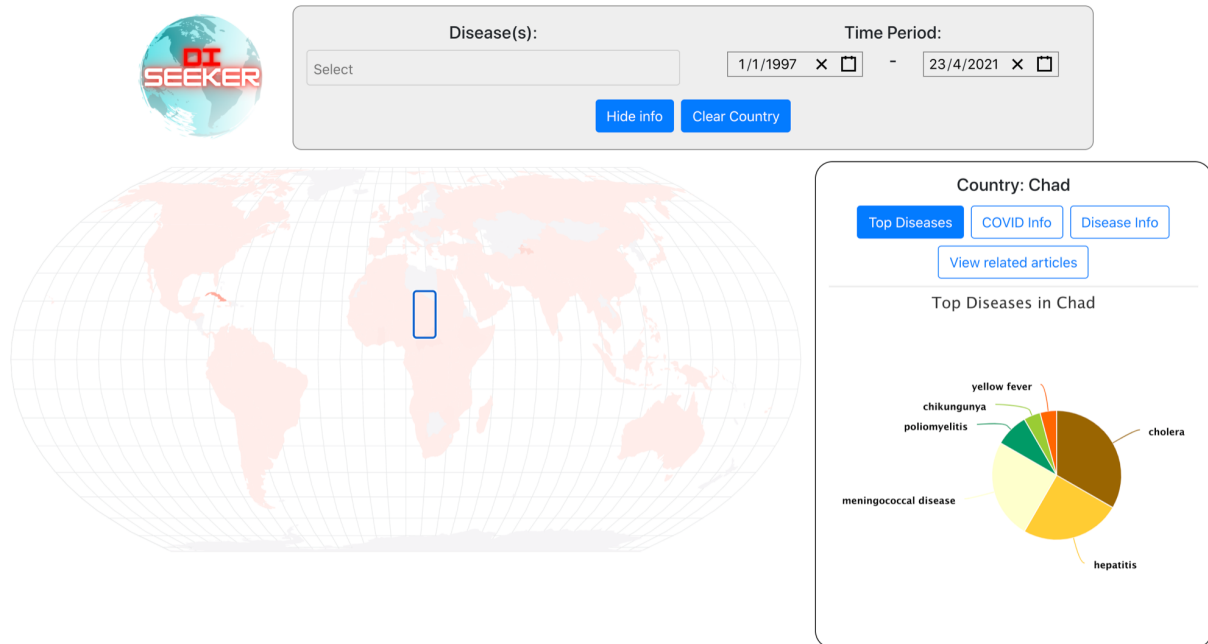


Figure 4 - DiSeeker information panel with country selected

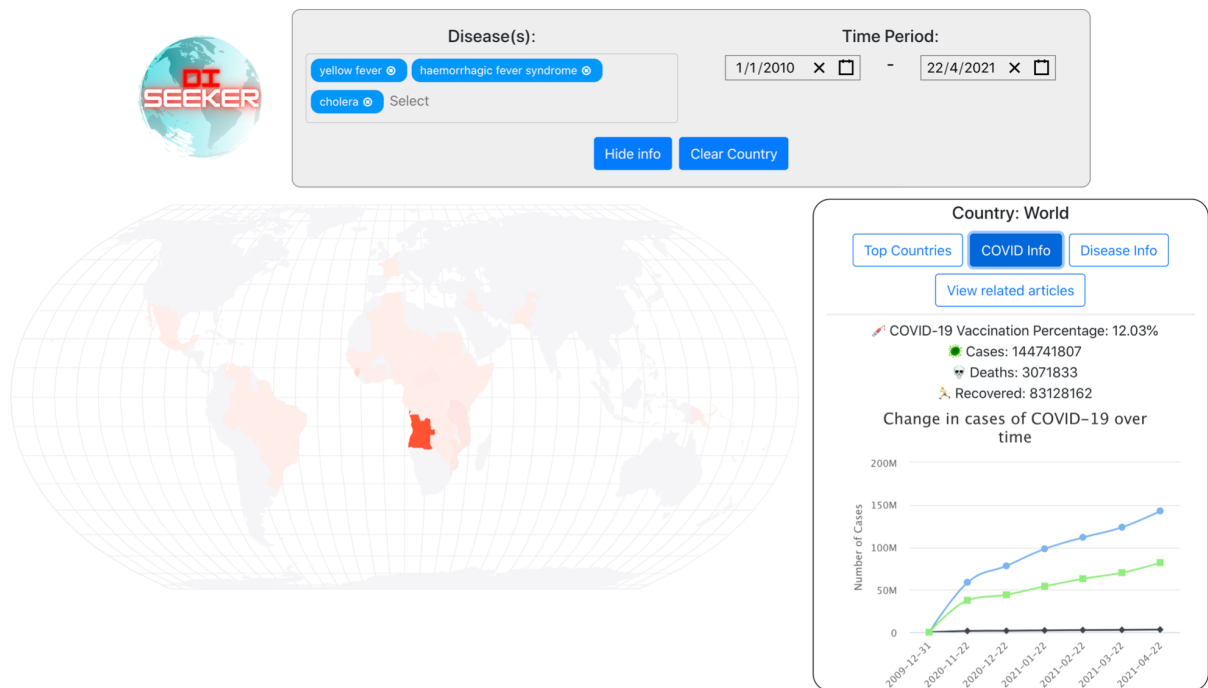


Figure 5 - DiSeeker COVID information panel

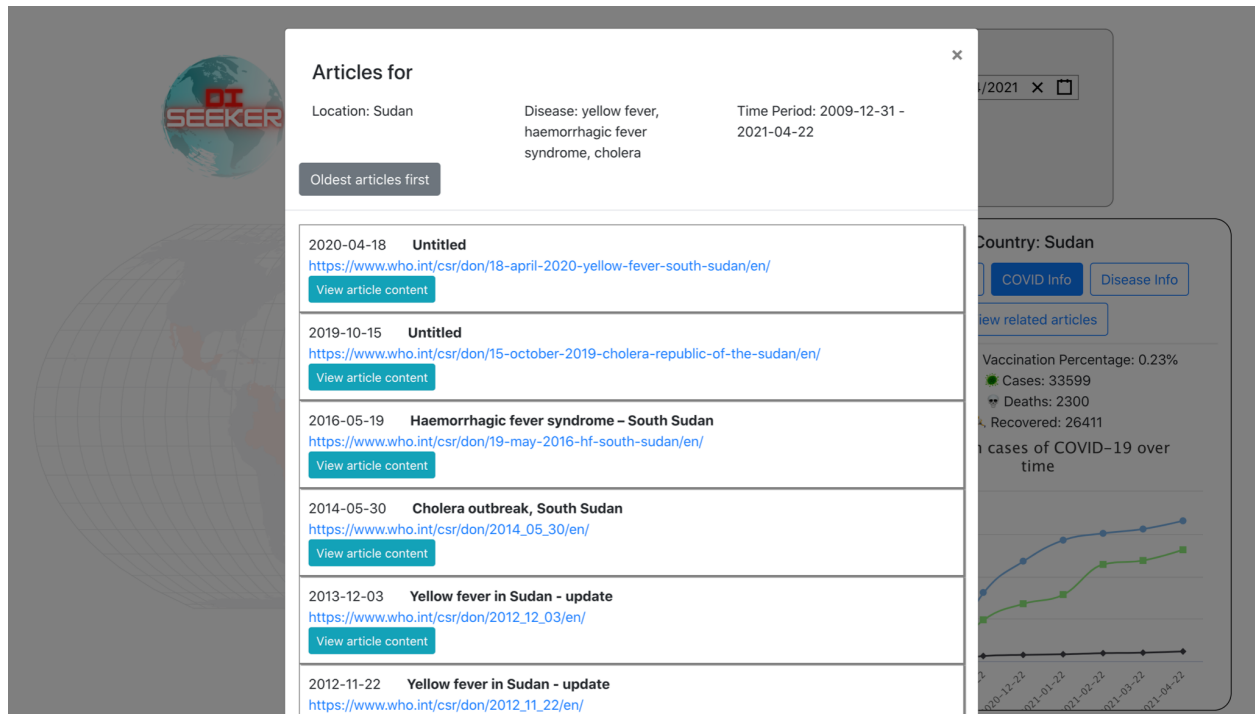


Figure 6 - DiSeeker articles popup

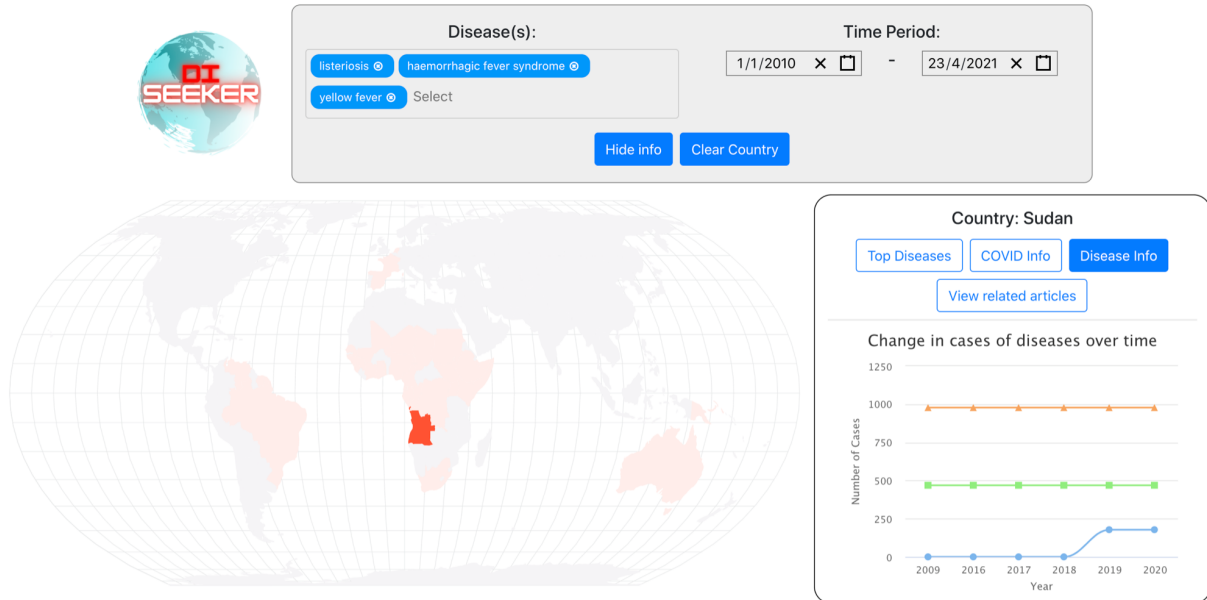


Figure 7 - DiSeeker information panel with case changes over time

Changes

Deliverable 2

- We're now using both Stoplight and Swagger for our API documentation. We've moved our API "Try It Out" functionality over to Swagger as we had issues accessing our AWS EC2 instance's static IP through Stoplight.
- Our lambda functions have now been replaced with a python script that is being run manually, we plan to automate this functionality later.
- We created a db_functions.py file that contains functions for retrieving relevant information from the database, formatting it into the appropriate json format and returning it to the flask server to send back to the user.
- We chose to use the Selenium python library for web scraping as opposed to BeautifulSoup

Deliverable 4

- We decided to go with ReactJS for our frontend
- We integrated 3 APIs into our project
 - Sour Dough API: For vaccination percentage statistics for each country
 - Brick Walls API: For Australian inter-state travel restriction information
 - Disease Sh API: COVID Cases, Recovered and Deaths numbers for our COVID information page (which included a graph of these statistics). Their data source for these statistics was John Hopkins University
 - Our Own Kool Kats API: For case numbers of many different diseases in most countries and their corresponding articles (all sourced from the World Health Organisation)