Software Engines Final Report

Executive Summary

The current EpiWatch disease outbreak surveillance system is extremely costly and inefficient, requiring lots of manual data collection on a regular basis. This is not only very time consuming but also very prone to mistakes. The report provides details on the development of Software Engine's disease outbreak surveillance and analytics platform; Outbreak Never (as an improvement of the current EpiWatch system).

Outbreak Never is an interactive, single page web application targeted towards medical professionals. It intends to assist medical professionals in their research, analysis and prevention of disease outbreaks by providing information on reports publicly accessible on the web.

Based on data collected from disease reports, the platform also provides suggested predictions for future disease outbreaks as a reference for medical professionals. Furthermore, as globalisation and travel plays a major role in disease spread and outbreaks, the system provides additional data on daily airplane routes to help the identification of potential routes of disease spread around the world and potential locations of future disease outbreaks.

All of the above is achieved through an automated procedure from the regular collection of data to its display and prediction.

Requirements

The following requirement has been prioritised using the MoSCoW notation;

[Must Have]- These outline the bare minimum required for a functional system.

[Should Have]- These are requirements that extend the basic functionality of the system.

[Could Have]- These are requirements that improve the user experience through convenience and flexibility.

[Won’t Have]- These are requirements that are extensive to the core system and considerably more difficult to produce in our time-frame.

Requirements - API

1. The client is able to access all disease reports related to given search terms (optional) and location for a given a period of time (start time, end time) in JSON format. [R1] **[Must Have]**
2. The client is able to access all disease reports (With more details about them) related to given search terms (optional) and location (optional) for a given period of time (start time, end time) in JSON format. [R2] **[Could Have]**
3. The client is able to access logs in regards to details (incl. The API call, accessed time, the given inputs) of all calls made to the API in JSON format. [R3] **[Should Have]**
4. The client is able to access results of a prediction (for the following week) of the possible number of people who will be affected by the disease in the location for a given time, country and disease in JSON format. **[Could Have]**

Note: More details about the API can be found later in the report

Requirements - Outbreak Never Platform

Target Audience: Medical Professionals

Our platform consists of a dashboard of six different modules with different purposes and functionalities. The six modules we have are Search, Map, Airline Data, Warnings, Graphs and Table.

For each of the module, various features have been implemented:

- Search [R5]
  - Search disease reports by start date and end date only. **[Should Have]**
  - Search disease reports by start date, end date and location only. **[Should Have]**
  - Search disease reports by start date, end date and disease only. **[Should Have]**
  - Search disease reports by start date, end date, disease and location. **[Should Have]**
- Map [R6]
  - View locations of recent disease reports **(past fortnight by default)** marked on the map

  using circular markers **[Could Have] **
  - View the relative severity (number of people affected) of each disease reports on the map through the size of the circular markers for each report **[Could Have]**
  - View the type of disease for the reports on the map through colour-coded circular markers for each report **[Could Have]**
  - View the general details of the report (i.e. date, headline, a link to original article) when the corresponding circular marker is clicked on **[Could Have]**
  - View the names of the top 5 flight destinations on the map from the nearest airport of the clicked circular bubble location **(by default) [Could Have]**
  - View airline route(s) from nearest airport of a clicked circular marker to the top 5 flight destinations from this location on the map **(by default) [Could Have]**
  - View locations of recent disease reports **(between the specified date range) **marked on the map using circular markers when searched by start date and end date only. **[Could Have]**
  - View locations of recent disease reports **(between the specified date range in the specified location) **marked on the map using circular markers when searched by start date, end date and location only. **[Could Have]**
  - View locations of recent disease reports **(between the specified date range around the world relevant to the specified disease) **marked on the map using circular markers when searched by start date, end date and disease only. **[Could Have]**
  - View locations of recent disease reports **(of the specified disease between the specified date range in the specified location) **marked on the map using circular markers when searched by start date, end date, disease and location. **[Could Have]**
- Airline Data [R7]
  - View the top 5 flight destinations from the nearest airport to the clicked circular marker as written text when a circular marker is clicked on on the map, show routes on the map and data in flight data box **[Could Have]**
  - If location is entered, view the top 5 flight destination routes from a searched location on map and flight data box**[Could Have]**
  - If only key terms are entered, view the top 5 flight destinations for each disease associated with the key terms displayed in the flight data box **[Could Have]**
- Warnings [R8]
  - View the top 5 diseases with the largest increase in disease reports in past month with its percentage of increase in its reports as well the actual number of reports received **(by default) [Could Have]**
- Graphs [R9]
  - View the number of disease reports for each disease in the past month through a bar graph which shows the top 2 diseases and the remaining diseases as others. **(by default) [Could Have]**
  - View the number of disease reports for each disease in the past month through a line graph which shows the top 2 diseases and the remaining diseases as others. **(by default) [Could Have]**
- Table [R10]
  - View a summary of all disease reports from the past month**(by default)** as a table showing details including disease, time, country, location (city/state), type and headline for each report. **[Must Have]**
  - Sort alphabetically by each heading (|diseases, time, country, location, type, headline) of the table when the heading is clicked. **[Should Have]**
  - View the original report article when each headline of the table is clicked.**[Should Have]**

Use Cases

Use Cases - API

The assumptions for these uses cases are that:

1. The end user is connected to the World Wide Web
2. The cloud provider’s server is online

Also note that, the cloud provider may be offline for maintenance, which is out of our control

Use Case ID:	**1.1**
Use Case Name:	Articles API
Actor/s:	End User
Description:	The articles endpoint will be the main endpoint that users call. It will return a large json string containing information which will be specified to the project’s specification.
Preconditions:	None
Postconditions:	The end user is given a status code, request URL, response header and body
Priority:	1
Normal Course of Events:	The end user types in the three required fields, the start and end dates, and the location. Then the end user submits their request, then after a period of time, a response is received
Alternative Courses:	The optional keyterms field is sent. Then the end user submits their request, then after a period of time, a response is received
Exceptions:	Invalid data and/or no data is sent

Use Case ID:	**1.2**
Use Case Name:	Get logs from the API
Actor/s:	The end user
Description:	The articles/logs endpoint will be a supplementary endpoint that users call. It will return a large JSON string containing information about all the requests and responses handled by all the other APIs.
Preconditions:	None
Postconditions:	The end user is given a status code, request URL, response header and body
Priority:	1
Normal Course of Events:	The end user sends a GET request to API for log data. After a period of time, the API returns a response
Alternative Courses:	None
Exceptions:	None

Use Case ID:	**1.3**
Use Case Name:	Article Details API
Actor/s:	The end user
Description:	Call API with start date, end date, key terms, and location, but key terms and location are optional, also the returned data is more detailed and has a simpler structure
Preconditions:	None
Postconditions:	The end user is given a status code, request URL, response header and body
Priority:	2
Normal Course of Events:	The end user types in the two required fields, the start and end dates. Then the end user submits their request, then after a period of time, a response is received
Alternative Course 1:	The optional keyterms field is sent. Then the end user submits their request, then after a period of time, a response is received
Alternative Course 2:	The optional location field is sent. Then the end user submits their request, then after a period of time, a response is received
Alternative Course 3:	The optional keyterms and location fields are sent. Then the end user submits their request, then after a period of time, a response is received
Exceptions:	Invalid data and/or no data is sent

Use Cases - Outbreak Never Platform

The assumptions for these uses cases are that:

1. The end user is connected to the World Wide Web

2. The cloud provider’s server is online

Also note that, the cloud provider may be offline for maintenance, which is out of our control

Use Case ID:	**2.1**
Use Case Name:	Return report table based on search conditions
Actor/s:	The end user
Description:	The user searches with different fields and gets a table of reported outbreaks
Preconditions:	None
Postconditions:	Data is returned based on the fields sent and organised in a table
Priority:	1
Normal Course of Events:	The end user searches by entering data into the fields on the top of the page, and a table of data at the bottom of the page appears
Alternative Course 1:	User will be shown an alert box when there are no results
Alternative Course 2:	User can search by start and end dates only
Alternative Course 3:	User can search by start and end dates and key terms only
Alternative Course 4:	User can search by start and end dates and location only
Alternative Course 5:	User can search by start and end dates, location and key terms
Exceptions:	When invalid data is sent, an alert box will appear stating that there are no results

Use Case ID:	**2.2**
Use Case Name:	Display reported outbreaks on map
Actor/s:	The end user
Description:	For each reported outbreak returned for a search, display it on a Google Map API as a circle
Preconditions:	None
Postconditions:	Every reported outbreak appears on the map as a circle, each with a colour depending on the outbreak name, and a size relative to the number of people affected by the outbreak
Priority:	2
Normal Course of Events:	The end user searches by entering data into the fields on the top of the page, and then the map displays all the returned reported outbreaks on one Google Map API using a latitude and longitude of the outbreak
Alternative Course:	If no results are returned, the map does not have any circles on it
Exceptions:	None
Assumptions:	All reported outbreaks have an associated longitude and latitude

Use Case ID:	**2.3**
Use Case Name:	Display Flight routes on map by searching by location
Actor/s:	The end user
Description:	Depending on what has been searched, the map can display up to 5 airline routes on the Google Map API
Preconditions:	A location has been entered
Postconditions:	On the map, up to 5 lines from a location are shown representing the top destinations from that location
Priority:	2
Normal Course of Events:	The user searches for a location at the top of the page, then the map shows up to 5 routes for that location
Alternative Courses:	The user can also search with key terms, but the routes still rely on the location entered
Exceptions:	None

Use Case ID:	**2.4**
Use Case Name:	Show up to 5 routes on map after clicking on a reported outbreak
Actor/s:	The end user
Description:	When a user clicks on a reported outbreak on the map, up to 5 flight routes are shown on the map that represent the most common destinations for the closest airport at the location of that outbreak
Preconditions:	None
Postconditions:	Up to 5 flight routes are shown on the map coming from the clicked outbreak
Priority:	2
Normal Course of Events:	The user clicks on a reported outbreak on the map, up to 5 flight routes are shown on the map that represent the most common destinations for the location of that outbreak, and markers appear at those destinations, that can be clicked to show a label of the name of that destination
Alternative Courses:	Any outbreaks without a close airport will not show anything

Use Case ID:	**3.1**
Use Case Name:	Warnings
Actor/s:	The end user
Description:	The end user is probe with a table of data showing the top 5 largest increase of report in the past month
Preconditions:	4 Weeks of data in database
Postconditions:	Up to top 5 largest increase in report is displayed in the table with their name, number of report and percentage of increase.
Priority:	3
Normal Course of Events:	User are enters landing page and scrolls down to the middle of the page. The end user will see a table of analytic data that is green fonted.
Alternative Course 1:	User will be shown an alert box when there are no results
Alternative Course 2:	User can search by start and end dates only
Alternative Course 3:	User can search by start and end dates and key terms only
Alternative Course 4:	User can search by start and end dates and location only
Alternative Course 5:	User can search by start and end dates, location and key terms
Exceptions:	None

Use Case ID:	**4.1**
Use Case Name:	Graphing Data
Actor/s:	The End User
Description:	The end user is presented a graphical view of the data
Preconditions:	4 Weeks of data in database
Postconditions:	There are two side by side charts in total that is to presented created below the maps container. The first chart shows the top 2 diseases with the greatest amount of reports in a specified time through a bar graph and the second chart shows the changes (none/increase/decrease) of the number of reports of 7 most commonly occurring diseases in the past month via a line graph.
Priority:	2
Normal Course of Events:	When the user enters the landing page, the graphing functions will fetch the data from the API and load the graphical view of these data.
Alternative Course 1:	User will be shown an alert box when there are no results and the graph will be hidden
Alternative Course 2:	User can search by start and end dates only.
Alternative Course 3:	User can search by start and end dates and key terms only
Alternative Course 4:	User can search by start and end dates and location only
Alternative Course 5:	User can search by start and end dates, location and key terms
Exceptions:	None

Use Case ID:	**5.1**
Use Case Name:	Prediction
Actor/s:	The End User
Description:	The End User is presented an estimation of the number of people who are going to be affected by a particular disease in the next week. This is done using Linear Regression
Preconditions:	4 Weeks of data in database
Postconditions:	A table will about the prediction will be presented in a container next to warning’s table.
Priority:	2
Normal Course of Events:	The User Enters the landing page, a prediction data will be presented on a table next to the Warning table.
Alternative Course 1:	User will be shown an alert box when there are no results and the graph will be hidden
Alternative Course 2:	User can search by start and end dates only.
Alternative Course 3:	User can search by start and end dates and key terms only
Alternative Course 4:	User can search by start and end dates and location only
Alternative Course 5:	User can search by start and end dates, location and key terms
Exceptions:	None

System Design and Implementation

High Level System Design

API

Discussion of API Implementation Design Plans

Initially, there were only plans to make one API endpoint so during the design of our API, there were two API designs for our code implementation which our team had come up with. We contemplated on which one to implement and decided to implement a combination of Designs 1 and 2 for the one endpoint.

We implemented Design 2 using one endpoint/GET with the nesting methodology in Design 1 where rather than having 4 GET calls dependant on the data from another, we have internal functions which fetch data from the database in the nesting format. These functions make work delegation easier and ensure work is not repeated.

During implementation, we added three more endpoints since more API calls (see more details in API Specification section) were required to provide more data to the client. The nesting methodology in the interaction with the database was kept with all these new endpoints.

The comparison between these two designs are described in the table below:

	Design 1	Design 2
Description	This design separates the GET calls into 4 nested GET calls where the first GET call (article) will send a GET call to the second (reports). The second GET call will call the third GET call (eventreports) and so on. Each will return its own JSON object.	This design implements only one GET call with methods in the code that simply provides data to create this one big JSON object.
Pros	This design allows the team to easily split the work between the members with a very strictly defined structure which avoids clashes and miscommunication.	This design is less abstract as it simplifies the logic where there is only one large JSON object to think about.
Cons	This design can be over-complicated as ultimately the user only needs one GET call to retrieve the data they would need. It would also be confusing as multiple endpoints/GET calls are exposed to the user despite the only GET call that they will be and can use will be the article GET call. This can be quite confusing for the user of our API.	This design is very ambiguous and difficult as to how the work would be split between the members in the team if it was simply decided to be this way. It could potentially lead to a lot of repeated work and miscommunication if it was not further defined to be clearer. Furthermore there is less freedom and will be very difficult to make adjustments to the code later on.

API Specification (Final Version)

After some evaluation of the usefulness of our previous designs from the perspective of an user, this is our current implementation of the API.

HTTP method	**Path**	**Parameters**	**Response**	**Auth**	**Description**
GET Call for the Users of API					
Get	/articles	start time, end time, keyterms, location	JSON Responses	No	User will call this HTTP method. Called by user.
Get	/articles/logs	N/A	JSON Responses	No	User will call this HTTP method. Called by user.
Get	/articles/details	N/A	JSON Responses	No	User will call this HTTP method. Called by user.

Get /articles/

Description

The articles endpoint will be the main endpoint that users call. It will return a large json string containing information which will be specified below.

Input

Example Request: http://46.101.167.163:5100/articles/?start%20time=2009-01-17T22%3A41%3A00&end%20time=2019-03-30T22%3A41%3A00&keyterms=swine&location=india

Parameter	Object Type	Description
StartTime*	Datetime Object	The starting date, all reports will be after this date
EndTime*	Datetime Object	The ending date, all reports will be before this date
KeyTerms	String	The key terms to be searched
Location*	String	Country of the outbreak/report

Output

Parameter	Object Type	Description
URL	String	Webpage this information is from
Date of publication	String	Date in which webpage was published
Headline	String	Title of webpage
Main Text	String	Text found in webpage
Report	Report Object	List of report objects

E.g.

[

{

```
"url": "<string>",

 "date_of_publication": "<string::date>",

"headline": "<string>",

"main_text": "<string>",

"reports": "[<object::report>]"
```

}

]

**NOTE: A list of article objects that match the user’s input will be returned to the user

Get /articles/details

Description

The articles/details endpoint will be another endpoint that users call. It will return a large json string containing information which will be specified below.

The difference between the articles endpoint and this endpoint is that this endpoint allows the location input parameter to be optional and returns some extra information such as a more details url, longitude and latitude of reported locations.

*The need for an endpoint like this resulted from assessing its usefulness from a user's perspective.

Input

Example Request: http://46.101.167.163:5100/articles/details?start%20time=2009-01-17T22%3A41%3A00&end%20time=2019-03-30T22%3A41%3A00&keyterms=swine&location=india

Parameter	**Object Type**	**Description**
StartTime*	Datetime Object	The starting date, all reports will be after this date
EndTime*	Datetime Object	The ending date, all reports will be before this date
KeyTerms	String	The key terms to be searched
Location	String	Country of the outbreak/report

Output

Parameter	**Object Type**	**Description**
URL	String	Webpage this information is from
Details URL	String	Another link that gives more information about this report
Date of publication	String	Date in which webpage was published
Headline	String	Title of webpage
Main Text	String	Text found in webpage
Disease	String	List of Strings of disease(s) involved in the article
Syndrome	String	List of Strings of syndrome(s) stated in the article
Type	String	The type of event in the article (can be one of: "presence", "death", "infected", "hospitalised", "recovered"
Date	String	Date in which the article event happened
Country	String	Country in which the article event happened
Location	String	City/state in which the article event happened
Latitude	Float	Latitude of the location in which the article event happened
Longitude	Float	Longitude of the location in which the article event happened
Number Affected	Integer	Number of people affected by the article event
Comment	String	Any comment in regards to the article event

E.g.

[

{

```
"url": "<string>",

"details_url": "<string>",

"date_of_publication": "<string::date>",

"headline": "<string>",

"main_text": "<string>",

"disease": [

  "<string::disease>"

],

"syndrome": [

  "<string::syndrome>"

],

"type": "<string::event-type>",

"date": "<string::date>",

"country": "<string>",

"location": "<string>",

"latitude": <float>,

"longitude": <float>,

"number-affected": <integer>,

"comment": "<string>"
```

}

]

**NOTE: A list of article details objects that match the user’s input will be returned to the user

Get /articles/logs/

Description

The articles/logs endpoint will be a supplementary endpoint that users call. It will return a large json string containing information which will be specified below.

Input

Example Request: http://46.101.167.163:5100/articles/logs

No input parameters are required to make this call.

Output

Parameter	Object Type	Description
Logs	String	The content of the logfile as a string for data about users who have accessed this API in history.

E.g.

{

```
"logs": "Global Incident Map : Team SoftwareEngines,
Accessed Time: 2019-04-01 14:34:01 GET SUCCESS
2019-02-20 00:18:00 to 2019-03-30 22:41:00 Location Taiwan\nKey terms [fever]\n\n"
```

}

Get /articles/prediction

Description

The articles/prediction endpoint will be another endpoint that users call. It will return a json string containing information which will be specified below.

Input

Example Request: http://46.101.167.163:5100/articles/prediction?date=2019-02-17&country=India&disease=Swine%20Flu

Parameter	Object Type	Description
Date*	String (yyyy-MM-dd)	The starting date for the prediction
Country*	String	The country to predict for
Disease*	String	The disease to predict for

Output

Parameter	Object Type	Description
Prediction	Float	The predicted number of people affected

E.g.

{

```
"prediction": 360.0
```

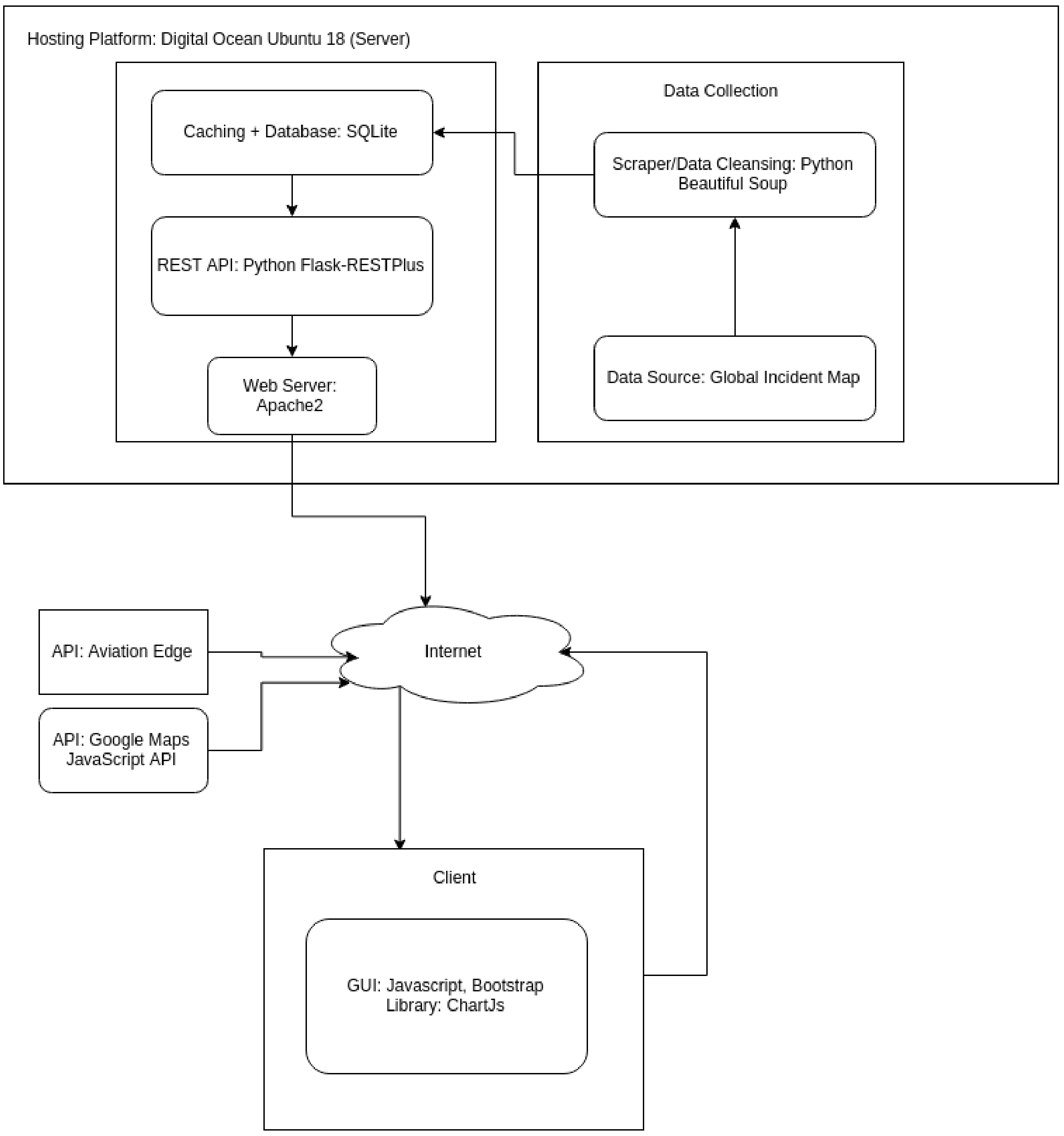
}

Outbreak Never Platform

Please refer to the appendix for screenshots of the Outbreak Never platform. {#please-refer-to-the-appendix-for-screenshots-of-the-outbreak-never-platform}

Concrete Implementation

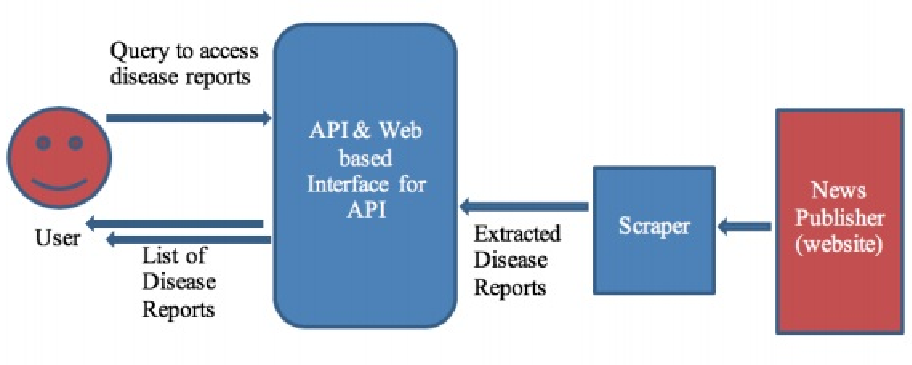
Final Software Architecture (API and Outbreak Never Platform)


Hosting Platform: Digital Ocean Ubuntu 18 (Server)
Caching + Database: SQLite
REST API: Python Flask-RESTPlus
Web Server: Apache2
Data Collection
Scraper/Data Cleansing: Python Beautiful Soup
Data Source: Global Incident Map
Internet
API: Aviation Edge
API: Google Maps JavaScript API
Client
GUI: Javascript, Bootstrap Library: ChartJs

As seen in the above image of our software architecture, we use our web scraper written in Python with BeautifulSoup to regularly scrape data from Global Incident Map. This data gets cleansed and placed in our database. We retrieve the data from our database and format it to supply it to our API users.

Our Outbreak Never client makes API calls to fetch JSON formatted data from the API which gets processed and displayed on the front end. Furthermore, the Outbreak Never platform also fetches airline data from the Aviation Edge API to use. It also uses the Google Maps Javascript API to implement the map functionalities on the front end.

Our API structure design follows the structure specified in the Project Specification:


Query to access disease reports
API & Web based Interface for API
User
List of Disease Reports
Extracted Disease Reports
Scraper
News Publisher (website)

In general, majority of our stack have been decided based on the idea of efficiency and small learning hurdle as there were only 10 weeks to complete the entire project so we were very short on time to produce quality software.

Besides this, the team agreed that the integration of different components individuals and sub-teams will be working on could potentially have a lot of work/issues that need to be resolved, so we decided against using tools that we are too unfamiliar with to prevent unnecessary stress under the tight schedule of trimesters.

Ideally if time was not a constraint, there are many new tools we would like to learn to use and experiment with.

Technology Stack

Back End

Database: SQLite3

For storing large amounts of complex data, such as the disease data scraped from the website and airport data, SQLite3 was chosen. This is due to a number of reason:

1. Many of our team members have previously had experience in using SQLite. This reduces the time taken to learn how to utilise a new tool so that we can focus on more major components of the project such as the API module itself with the short amount of time we have on our project. This allows the data scraper and API teams to need only agree on what schema was needed for the data instead of having to be concerned with how to store data from the scraper and retrieve the data for the API.
2. SQLite is simple and lightweight which perfectly suits the purpose of our project where there is not too much variation in the data and the structure of data.
3. SQLite structures data using tables which is very convenient especially for our team and the data source we are scraping information from (Global Incident Map). Global Incident Map itself displays data in a table format so the process of retrieving the data from the table in Global Incident Map and placing it into our database tables will be straightforward. To easily provide the API with a suitable JSON format, the database schema will mimic the JSON formats we provide to the users of our API (as specified in the Project Specifications).

Our team considered NoSQL databases such a MongoDB which can provide a very flexible and dynamic structure with data stored like JSON-format documents. This could potentially make it easier for the team to process and provide the data to users of our API as what we require to return in our API are JSON formats. However, it may potentially complicate the web scraping and storing of data from Global Incident Map. On top of this, most of our team members have never had experiences in NoSQL databases so this could be a very steep learning curve that could take up a lot of our team. Hence our team decided on SQLite as our database.

Development Language & Library Usage: Python and Beautiful Soup 4

Due to the flexibility and our team's familiarity with Python, our team has decided to use this language to implement our API module. This will reduce our learning curve by removing the time taken to learn a new language especially when most of the team already has a steep learning curve about the structure of an API. Furthermore, due to Python's popularity, there are a lot of easily-accessible resources online which would greatly assist us in creating our API. Furthermore, due to Python's popularity, there are a lot of easily-accessible resources online which would greatly assist us in creating our API.

Other languages our team members are somewhat familiar with such as Java and Javascript are possible options. Java is a typed-language and not dynamic like Python which could potentially create more hurdles for us than necessary if we were to use it. Javascript is a very powerful and flexible language with most members being experienced in it. However, because in our team, we have one member who is experience with developing APIs in Python, we have decided to use Python to effectively utilise every team members' knowledge and skills sets.

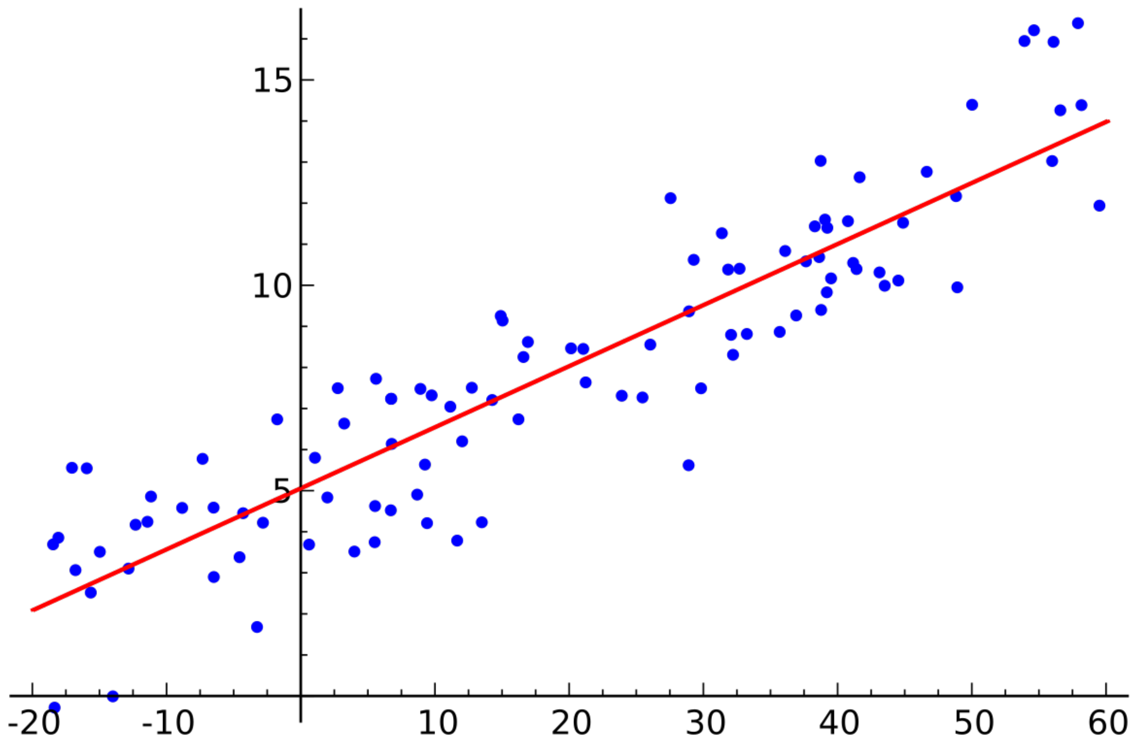
For web scraping data from our data source, some of our team members have had experience in web scraping with Python Beautiful Soup. Others have had some experience in web scraping with Perl. To keep the process simple and efficient so that we can focus on other more significant parts of the project, we have decided to use Python Beautiful Soup.

Framework usage: Flask-RESTplus, Pandas & Scikit-learn

As mentioned before, one of our members have had experience with Flask-RESTplus and they are comfortable using it again due to our familiarity with it. The rest of our team believes that using a framework which someone in the team is already familiar with will not only reduce the number of learning hurdles but also allow our members to comfortably learn to implement an API under the guidance of the experienced team member.

We have incorporated a prediction endpoint in our API, in which we use both pandas and scikit-learn modules to make a prediction on how many people will be affected for a disease, location and date. We use pandas DataFrames as a means of easy data manipulation and analysis (e.g. data cleansing). We use Scikit-learn Linear Regression model to predict the number of people affected.

Summary of how we predict - We convert our data points and graph them, draw a regression line in which we use to predict the number of people affected.



Development and Deployment Environment

Environment: Digital Ocean, Linux & Apache2

Web Browser: Firefox

We came across Digital Ocean because of a recommendation by our mentor. We had other alternatives like Heroku and AWS, but in the end we chose DigitalOcean because of its reliable and easy to use nature. Digital Ocean's focus on user experience and great documentation and community simplifies deployment for beginners like us. DigitalOcean also has a relatively low entry level cost which is ideal a for low budget project like ours. Although we are still beginners when it comes to deploying software, we still want to have some flexibility on choosing the underneath architecture of our platform and Digital Ocean complements this need of ours. This is because it is a type of Infrastructure as a Service (IaaS). This means that DigitalOcean only provides the server and will actively operate our API if we wish to deploy on it. This feature provides great flexibility for us to choose which other software to setup, interact, maintain and deploy our code onto DigitalOcean. Unlike other alternatives like Heroku and Microsoft Azure that is a Platform as a Service (PaaS) which provides a fixed platform to run and manage. This is not as flexible as IaaS since PaaS does not give us the freedom to manage our infrastructure.

Apache2

For our web server, we wanted to stick with our LAMP stack structure. We also have team members who have experience with NGINX and Apache2. Recalling that our focus is to have a smooth learning curve, our choices were soon came down to both of these web servers. In the end, we chose Apache2 because Apache2 has better support for LAMP structure and is recommended by DigitalOcean's community and a well documented step-by-step guide of how to get things started was published on their forums… Apache2 also offers dynamic more official modules which adds the flexible perspective into our stack than NGINX.

Front End

Template Used: https://designrevision.com/downloads/shards-dashboard-lite/

JavaScript/JQuery/Node.js/ChartJs

The frontend of the site is coded in JavaScript. For the frontend of the website, the JQuery library was used to simplify the manipulation of HTML elements on the webpage as well as to make requests to the server sending airport data responses. JavaScript was chosen for this task due to the familiarity team members had with it, as well as the fact that there is a large amount of documentation and support for this language. ChartJs is used to display our graphical view of data. This is used because the original template uses ChartJs and it makes it easier to create charts on top.

The server that analyses flight data from an external API and serves the analysed data to the frontend of the website is coded in JavaScript and uses the Node.js environment. This is as it allows access to local files in the server directory and team members also have experience with it.

Additional Third-Party Data/APIs Used (on top of Global Incident Map)

Flight Data API

For our flight analytics, we used the Aviation Edge API as a source of flight data. The two types of data that were retrieved from this API were airport data, such as the name, IATA code, longitude and latitude coordinates, and information of flight routes from a given airport, which included the IATA codes of the origin and destination airports.

Google Maps JavaScript API

In order to map out the reported locations of the disease outbreak reports as well the the different airline routes, we used the Google Maps Javascript API. We chose Google Maps over other web mapping libraries such as Polymaps because it is very well known and one of our members has already had experiences in using it.

Summary of Key Benefits/Achievements

1. Our API attempts to provide more modularity, and is designed to be easy to manage and maintain. This makes it simple for us to extend on our API and develop additional endpoints if needed.
2. The SQLite Database allows for concurrent reading and sequential writing, which is extremely useful for our project in this case. It makes use of multi CPU which is beneficial to us as it makes reading and querying information from the database faster. Also, the sequential write does not negatively affect our database greatly as we do not write into the database often, only once an hour.
3. Many of the choices on programming languages to use were based on familiarity as we decided that it was most important to use languages that we were most familiar with. This allowed us to minimise the learning curve for this project which decreased the average implementation time for each feature. As a result, we were able to maximise the amount of key features implemented, which resulted in a more functional and appealing application.

Team Organisation and Conclusion/Appraisal of Work

Responsibilities/Organisation of the team

At the start of the term, we had a discussion on the responsibilities from every member on the team. We outlined each of the team member’s strength and weakness, preferences and their commitment to the course. And then coming up with the arrangement below. However, as the term goes, we quickly adjusted our focuses for Phrase 2 as the requirements are different from Phrase 1 as shown below. During the transition into Phrase 2, in addition to our allocated work and responsibilities, every team members had to research and find out how prediction works in the medical industry, what kind of diseases are relevant and what are the major factors of diseases spreading globally.

Team Work Arrangement and Responsibilities (Phase 1)

Andy mainly be designed, created the skeleton structure of the API, tested and oversaw the whole process of developing the API. He is the member with the most theoretical knowledge and practical experience in making APIs so he took on a lead role in the API component of the project to greatly guide our team in working towards a clearer goal, completing our tasks more efficiently. Due to this, Andy was also assigned as our project manager in our agile methodology that we adopted and also worked on coding up the backend logic for the client side of the project (the website).

Tammy assisted Andy in designing and developing the API module as a new learning experience. She also be wrote up the documentation for our code due to her previous experiences with documentations through various group projects where she documented, structured and formatted reports and codes.

Wilson helped with coding the API and creating the database queries for the API module to access our database as a new learning experience.

He also worked on developing logic of the client side (website) for this project due to his familiarity with backend coding through other similar projects. In scrum terminology, Wilson was the SCRUM Master overseeing the work in each sprint.

Jacky created/set up the database to store all the necessary scraped information due to his familiarity with databases. He also helped Marshall with the web scraping and data cleansing part of the project.

Marshall was tasked with developing the web scraping and data cleansing for this project as he has had prior experience and knowledge in web scraping. He also assisted Jacky in creating the database to store all the needed web-scraped data.

Team Work Arrangement and Responsibilities (Phase 2)

Andy worked on connecting the end points and developing the backend. For the frontend, he worked on implementing one of the key display features for our application, a tab that predicts the number of people affected by a particular disease in the coming week using machine learning. He also collaborated with Marshall in creating an interactive table listing disease outbreaks which can be sorted and filtered.

Tammy developed the map features with Google Maps API as she has some past experience developing an application with Google Maps. Tammy was also responsible to work closely with Marshall to integrate flight data into the application, which also links dynamically with her map features. With some experiences and passion in designing and producing user-friendly user interfaces for similar projects, she took lead in facilitating the creation of user stories for the client side of the project.

Wilson helped the team reformat all their front end code to seamlessly fit all our components into our front end template, and also worked on making the website more visually appealing through CSS. Due to his inexperience with JavaScript, he was not tasked to complete any key features for the application but still made a simple analytics feature which lists diseases that have had a major increase in reports in the past month. He also worked with Jacky and Marshall on the presentation side of the project.

Jacky worked on the frontend of the client side for this project as he has had experience in frontend coding from previous software projects. This included implementing features such as the bar graph and line graph using Chart.js which displays the most commonly reported diseases globally or for a particular location, depending on the search terms. He also worked with Wilson and Marshall on the presentation side of the project.

Marshall was mainly tasked with the integration of Aviation Edge API by scraping and caching the data in a database. He also worked closely with Tammy to ensure that the flight data is displayed correctly in two of the key frontend features: the flight table and the map view. He also worked with Andy to create the interactive table listing the outbreaks and with Wilson and Jacky on the presentations for the project.

Project Reflection

Major Achievements

1. From our extensive research, we were able to conclude that globalisation was recognised by medical professionals as a key factor in the increasing global widespread of diseases. With this knowledge, we decided to scrape and provide flight information on any location globally to try aid these professionals and this is our proudest feature of the application.
2. We decided on using programming languages we were most familiar with in order to create a smoother learning curve in the project. This helped us greatly as we were able to allocate more time to researching and developing more features in a specified time, resulting in a website with a plethora of features in which we are proud of.
3. We also wanted to create the minimum amount of confusion for people using our website. We did not wish for users to not know where to click when on our application, so we decided to create a dynamic Single-Page Application that was extremely simple to navigate through, with nothing but

a search bar that helps the user find whatever information they desire.

Issues/Problems Encountered

Web Scraping

One of the challenges we encountered was dealing with dynamic data during web scraping. Our source website had some of its data dynamic generated using JavaScript. This caused some inconvenience to us since our script were made for statically generated data. We brought this issue up to our mentor's attention and his advice was to leave some of those data out as it can be very challenging. As a result of this, some of the data would not be available in our database.

It would have helped us greatly if we were more knowledgeable on how to scrape dynamically generated data from a website. This would have saved us time as we spent a much longer time than was allocated on the web scraping part of the project due to our inexperience. Fortunately, we were able to compensate for this by completing most of the other parts of the project ahead of schedule.

API Implementation and Testing

One of the challenges in the API implementation was integrating the API code with the database query code. To prevent too many issues during integration, we strictly defined the interface in between specifying details about the data type as well as the format to be returned.

Another issue that arose in regards to the API was during the process of testing. Test cases that were written for black box testing for the API expected different outputs to what the implementation of the API was producing. Rather than this being an issue of the API not producing the correct output, it was more of an issue of miscommunication as we did not strictly define what kind of response different types of behaviour should elicit, what was acceptable and what was not acceptable. This problem led us to having to adjust accordingly our test cases as well as our API code to align the expectations for the outputs and behaviours later on.

Code Quality

Code quality and readability was an issue that arose within the team. This affected the efficiency of our work since members had to take longer to understand one another's code. A good quality will make other people other than the writer quickly understand what the code is doing and sometimes understand why certain code style is used instead. When the code quality is high, any modification to the code wouldn't break other parts of the code and repeated coding work can be very easily avoided.

In our team, we have members who are not used to certain languages or did not have as much experiences compare to other team members, and ultimately causing some of the code quality to be poor. Furthermore, due to the fact that the time we are given is much shorted and limited to previous term, many of our members were leaning to writing code that works to meet the deadlines rather than spending the time to have a proper design and communicate with other members to write good code. As a result of this, our team felt the bad effects of bad code quality and caused us to have a harder time to enhance, modify our features. However, we took the time to redesign some of our codes so that we can make the important changes to present our features.

We think that writing good code and reading codes from a large piece of software is a skill and requires practice. However, we do wished that we had workshops to improve our code quality since software engineers often work together as a team. A suggestion to this would be having us to work on an open source project. This is beneficial as many of today's large industry and company's software engineers work on existing code base.

Front-end

Another skill that would have been useful to us is the ability to develop single-page applications. We all had previous experience using javascript for this but the experience was relatively shallow. As a result, we encountered many problems when developing our frontend but we were able to overcome these in the end.

Reflection on Teamwork

Overall, our teamwork went smoothly. Regular (weekly) meetings were held to update each other, discuss our ideas and work together. This allowed us to plan as a team to identify what can be done in the near future, Our work is also carefully delegated to every member in the team so that we can maximise our strengths and minimise our weaknesses. This is based on skills and abilities, preferences from each member. Furthermore, we had a SCRUM master who keep tracks of work from every member. This benefited a lot on making sure work was done at each stage of the project. Finally, the whole project is proudly contributed by everyone and there is no unbalance of workload between team members.

On the other hand, there are still some flaws in our approach. If we were to work on the project again, one thing that we would like to do differently is communicate with each other more regularly outside of meetings. This would allow the team to address small questions, mistakes together immediately, preventing potential future problems.

Additionally, much of our time was spent on developing the API due to our lack of experience and losing track of our goals. It would have been better if we divided the time equally to front end and back end and started the front end planning and designing earlier. This would not only reduce the stress on every team member but also allow us to add more features to our front end and further develop what we currently have with improved code quality.

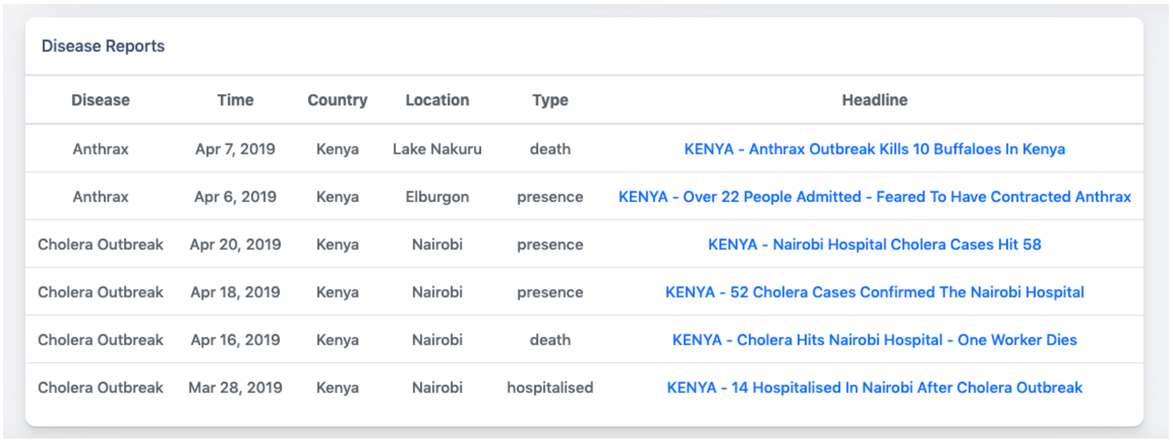
Appendix

Definition(s)

Phrase	Definition
Datetime Object	yyyy-mm-ddThh:mm:ss

Screenshots

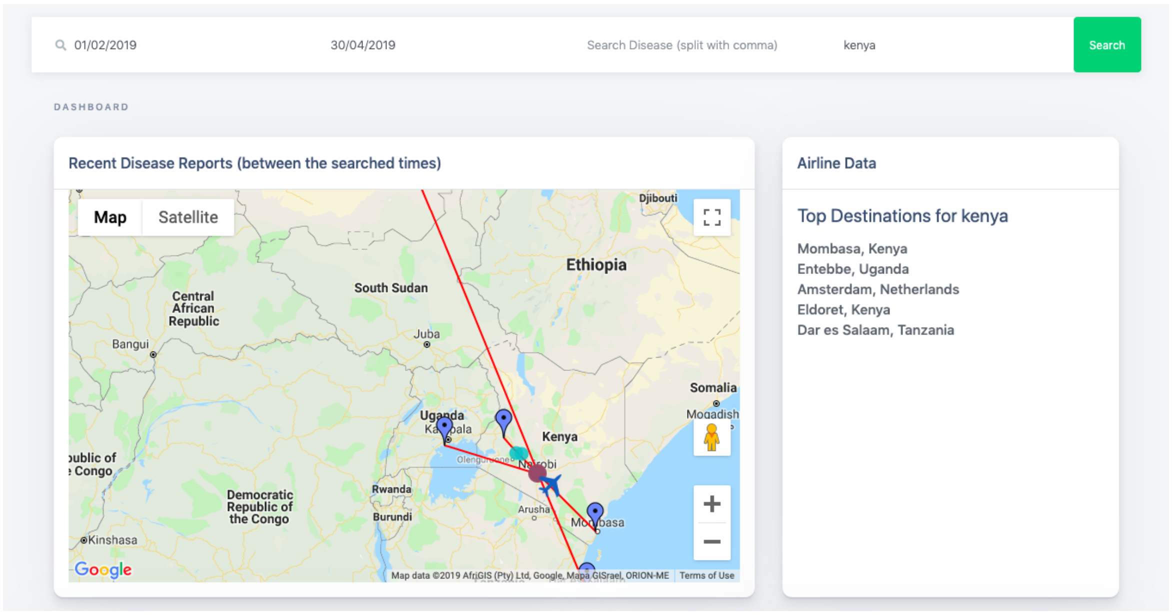
Table of reported outbreaks



Disease Reports

Disease	Time	Country	Location	Type	Headline
Anthrax	Apr 7, 2019	Kenya	Lake Nakuru	death	KENYA - Anthrax Outbreak Kills 10 Buffaloes In Kenya
Anthrax	Apr 6, 2019	Kenya	Elburgon	presence	KENYA - Over 22 People Admitted - Feared To Have Contracted Anthrax
Cholera Outbreak	Apr 20, 2019	Kenya	Nairobi	presence	KENYA - Nairobi Hospital Cholera Cases Hit 58
Cholera Outbreak	Apr 18, 2019	Kenya	Nairobi	presence	KENYA - 52 Cholera Cases Confirmed The Nairobi Hospital
Cholera Outbreak	Apr 16, 2019	Kenya	Nairobi	death	KENYA - Cholera Hits Nairobi Hospital - One Worker Dies
Cholera Outbreak	Mar 28, 2019	Kenya	Nairobi	hospitalised	KENYA - 14 Hospitalised In Nairobi After Cholera Outbreak

Flight routes from clicked outbreak


01/02/2019 30/04/2019 Search Disease (split with comma) kenya Search

DASHBOARD

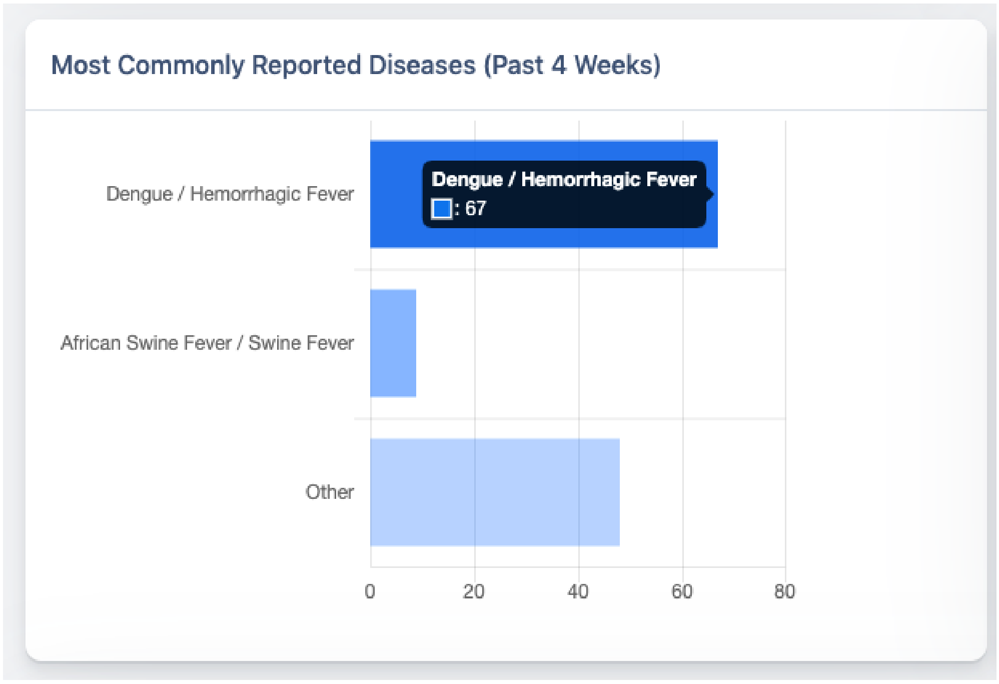
Recent Disease Reports (between the searched times)

Airline Data

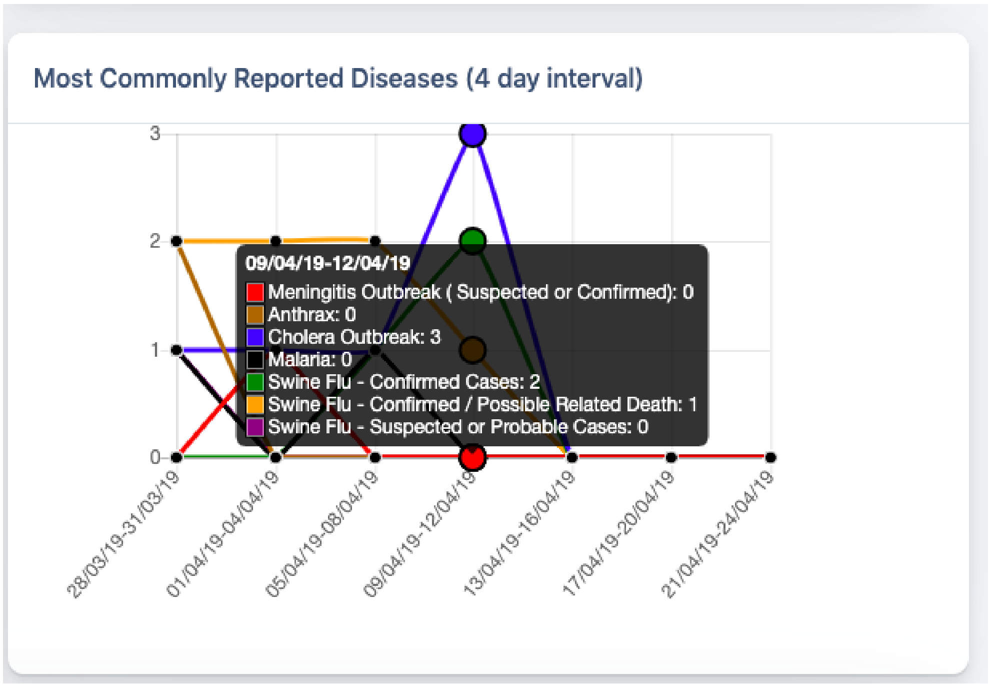
Top Destinations for kenya

Mombasa, Kenya
Entebbe, Uganda
Amsterdam, Netherlands
Eldoret, Kenya
Dar es Salaam, Tanzania

Bar Graph Data for total outbreaks of a disease

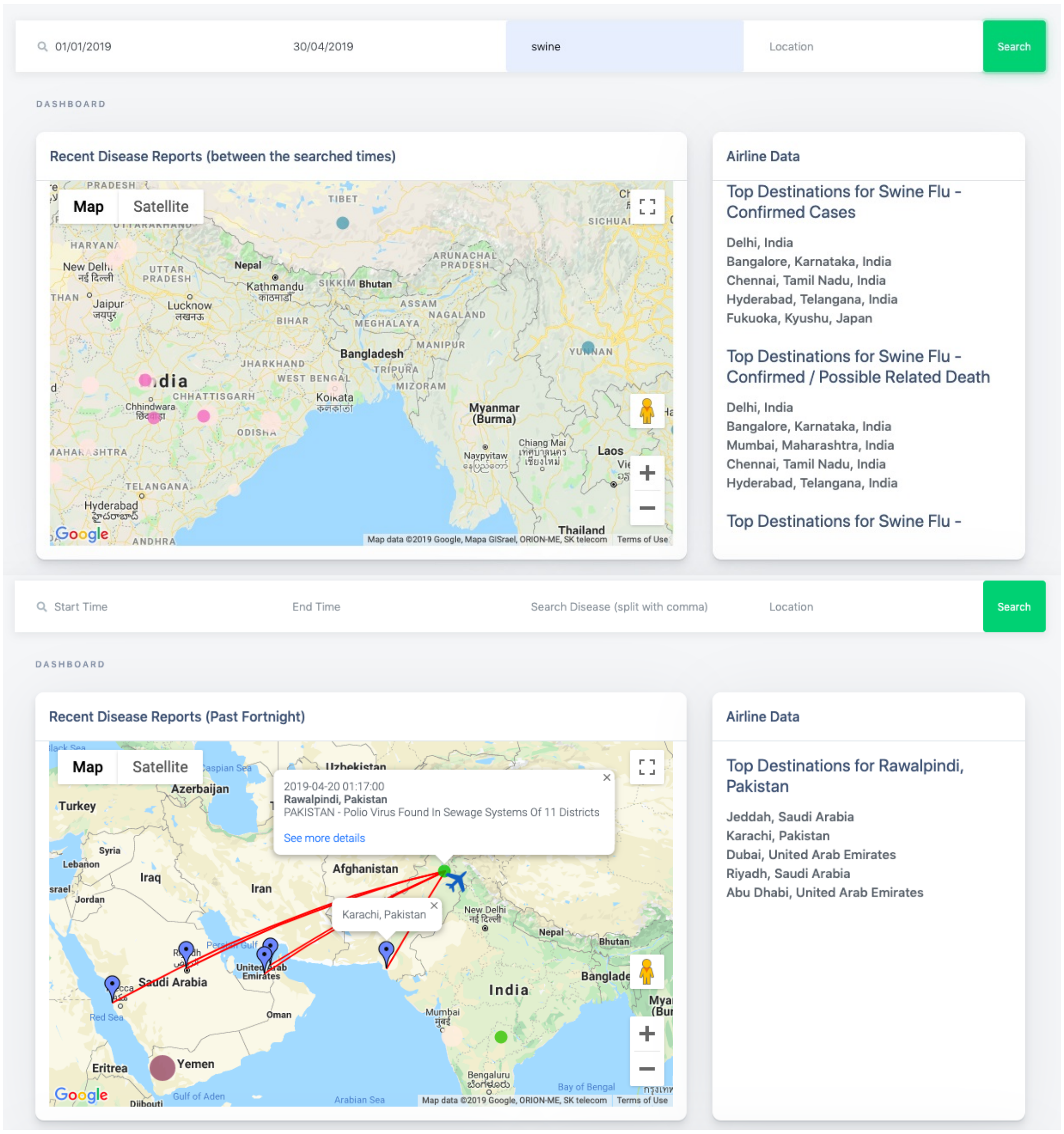

Most Commonly Reported Diseases (Past 4 Weeks)

Line Graph Data for trends


Most Commonly Reported Diseases (4 day interval)

09/04/19-12/04/19
🟥 Meningitis Outbreak (Suspected or Confirmed): 0
🟫 Anthrax: 0
🟦 Cholera Outbreak: 3
⬛ Malaria: 0
🟩 Swine Flu - Confirmed Cases: 2
🟧 Swine Flu - Confirmed / Possible Related Death: 1
🟪 Swine Flu - Suspected or Probable Cases: 0

Search with date range and location


01/01/2019 | 30/04/2019 | swine | Location | Search

DASHBOARD

Recent Disease Reports (between the searched times)

Airline Data

Top Destinations for Swine Flu - Confirmed Cases

Delhi, India
Bangalore, Karnataka, India
Chennai, Tamil Nadu, India
Hyderabad, Telangana, India
Fukuoka, Kyushu, Japan

Top Destinations for Swine Flu - Confirmed / Possible Related Death

Delhi, India
Bangalore, Karnataka, India
Mumbai, Maharashtra, India
Chennai, Tamil Nadu, India
Hyderabad, Telangana, India

Top Destinations for Swine Flu -

Start Time | End Time | Search Disease (split with comma) | Location | Search

DASHBOARD

Recent Disease Reports (Past Fortnight)

2019-04-20 01:17:00
Rawalpindi, Pakistan
PAKISTAN - Polio Virus Found In Sewage Systems Of 11 Districts
See more details

Karachi, Pakistan

Airline Data

Top Destinations for Rawalpindi, Pakistan

Jeddah, Saudi Arabia
Karachi, Pakistan
Dubai, United Arab Emirates
Riyadh, Saudi Arabia
Abu Dhabi, United Arab Emirates