Team we-need-REST

Final Report - EpiWatch Epidemic Tracker

Requirements

Upon beginning the project, we were given the following functions that the end user should be able to perform using our application:

- Ability to integrate data from different sources and present them in a user-friendly way (1)
- Ability to browse news related to a disease outbreak over a period of time/geographically and identify news about outbreak of interest (2)
- Ability to examine social media related posts on disease outbreaks over a period of time and identify particular trends (3)
- Ability to provide advanced analysis facilities such as analysing the impact of an outbreak on residents of the region over a period of time or predicting potential outbreaks based on historical patterns (4)

Within our software solution, we have chosen to address these requirements in the following ways:

Feature	Application implementation
(1)	Using our web scraper, we are able to obtain data from our specified source (Global Incident Map) without any difficult user input. This information is collected into a JSON file by the web scraper, eliminating the need to manually find and input data. The specified source (Global Incident Map) contains the latest reports from a wide variety of websites from different origins and publications, making it an ideal source of information for us to collect from.
(2)	All the data obtained from the primary source (GIM) is provided through our application, which enables the user to filter reports based on time, date, location and keyword (or a combination of these). This allows the user to perform a narrow search for precisely the outbreak they are looking for, or get results matching to their exact query.
(3)	Original plans for the application included integration with the Twitter API in order to examine tweets as part of the dataset we were working with. Unfortunately, this feature was not included in the final application due to time constraints not allowing for us to pursue this idea fully.
(4)	To satisfy this feature, we included both a small-scale and large-scale data analysis feature. In terms of small-scale data analysis, the 'summary' page provides the user with the ability to analyse a number of reports selected using the search feature from the main page, and collate a report which visualises all the reports onto a heat-map together. Additionally, the reports that are part of this summary are analysed using Natural Language Processing. This allows us to extract entities and relevance categories out of each report, giving a better idea of the contextual meaning of each report. For the large-scale data processing, our Machine Learning algorithm is able to analyse environmental and location data, as well as specialised data specific to the area or disease, to make predictions about the possibility of future outbreaks in a certain location. This allows researchers to better analyse large amounts of data by using the alrogithm to process it, instead of having to do lots of manual work to accurately make predictions.

In assessing the functions that needed to be completed, our team decided to break these down into the following requirements, which were completed across the various sprints within the agile process:

1. Backend API to scrape the Global Incident Map and return reports
    1. Obtain complete set of reports from Global Incident Map using web scraper
    2. Save all reports from web scraper into given JSON format for storage
    3. Data is accessed through a web-hosted API
2. Extend backend API functionality into a Graphical User Intereface
    1. Fetch a singular report
    2. Fetch all reports
    3. Filter by criteria
    4. Delete a report
    5. Create a report
    6. Update/Edit a report
3. Summarise selected data in 3 ways:
    1. Display list of collected reports
    2. Visualisation of location based data
    3. Natural Language Processing on main text within reports
4. Predict future outbreaks using ML
5. Collect relevant reports using criteria

We will continue to use these requirements as references for each use case. Anotated as a R followed by a digit on the top right of each use case card

Use Cases

1. View all reports **R1** **R2.2**

As a *User* I would like to be greeted with a list of the latest reports in order of recency
starts when I click on the homepage
ends when the page loads

Interactions:

- User accesses the website
- System directs them to hompage
- On load the page fetches data from API and presents list of latest reports in order of recency

---

Actors:

- User

Preconditions: User is on the Home page
Postconditions: A list of reports appears on the screen

2. Filter reports

R2.3

As a *User* I would like to Filter reports based on certain criteria
starts when I click on the homepage
ends when The search results are filtered appropriately

Interactions:

- User accesses the website
- System directs them to hompage
- User opens dropdown “search” modal and enters their chosen criteria
- System filters out and displays reports that satisfy these criteria

---

Actors:

- User

Preconditions: User is on the Home page
Postconditions: A list of filtered reports appears on the home page

3. Summary of multiple reports

R1 R2.1 R2.2 R2.3 R3

As a *User* I would like to view analytics on a collection of multiple reports, including geographical information and language analysis
starts when I click on the homepage
ends when The “Get Summary” page loads

Interactions:

- User accesses the website
- System directs them to hompage
- [Optional] User searches by criteria as in UC2
- User clicks on the reports they would like to analyse (or alternatively click the “Select All” button if they wish to see a more complete analysis)
- System adds selected reports to a list in preparation of summarising them
- User clicks “Get Summary”
- System redirects to a “Summary” page containing a list of all reports selected, allowing for further anaylysis on the page

---

Actors:

- User

Preconditions: User is on the Home page
Postconditions: A detailed summary of selected reports appears on-screen

3.1. Geographical Summary of Multiple Reports

R1 R2.1 R2.2 R2.3 R3.2

As a *User* I would like to view geographical information on a collection of multiple reports, in a clear visual format
starts when I click on the homepage
ends when The heat map appears

Interactions:

- User accesses the website
- System directs them to hompage
- [Optional] User searches by criteria as in UC2
- User clicks on the reports they would like to analyse (or alternatively click the “Select All” button if they wish to see a more complete analysis)
- System adds selected reports to a list in preparation of summarising them
- User clicks “Get Summary”
- System redirects to a “Summary” page containing a world map highlighting locations mentioned in the reports

---

Actors:

- User

Preconditions: User is on the Home page
Postconditions: A detailed heat map appears on the screen, highlighting all locations mentioned in the reports.

3.2. Natural Language Processing Summary of Multiple Reports

R1 R2.1 R2.2 R2.3 R3.3

As a *User* I would like to view a language-processing analysis on a collection of multiple reports.
starts when I click on the homepage
ends when The language-processing data appears

Interactions:

- User accesses the website

- System directs them to hompage
- [Optional] User searches by criteria as in UC2
- User clicks on the reports they would like to analyse (or alternatively click the “Select All” button if they wish to see a more complete analysis)
- System adds selected reports to a list in preparation of summarising them
- User clicks “Get Summary”
- System redirects to a “Summary” page containing a summary of the Natural Language Processing information extracted from each report

---

Actors:

- User

Preconditions: User is on the Home page
Postconditions: A list of NLP entities appear on the screen

4. Editing Report

R2.6

As a *User* I would like to edit the details of a report.
starts when I click on “Edit Report”
ends when I click “Submit”

Interactions:

- User clicks “Edit Report”
- System reveals a number of input boxes representing the different categories they may like to update
- User fills in the input boxes with the information they choose
- System overwrites the categories of the selected report with user’s inputs

---

Actors:

- User

Preconditions: User is Viewing Report (See UC2)
Postconditions: Some details of a report have been edited as required

5. Deleting Report

R2.4

As a *User* I would like to delete a selected report
starts when I select my report and am on the page where I can “View Report”

ends when I click “Delete”

Interactions:

- User clicks “Delete Report”
- System removes report so that it cannot be found
- System alerts the user that the report has been successfully deleted

---

Actors:

- User

Preconditions: User is Viewing Report (See UC2)
Postconditions: Report no longer exists in the database and cannot be found via the website

6. Creating Report

R2.5

As a *User* I would like to create a new report
starts when I click on the "Create" page
ends when I click “Submit”

Interactions:

- User fills out the list of input boxes for each category of their report
- System creates the report and assigns it a unique ID so it can now be found and analysed like all the others

---

Actors:

- User

Preconditions: User is on the “Create” page
Postconditions: New report with the information specified by user exists in the database

7. Predicting Outbreaks

R4

As a *User* I would like to predict where new outbreaks would appear
starts when I click on the “Predict” page
ends when Page and map have loaded

Interactions:

- User selects the “Predict Tab”
- System runs Machine Learning Algorithm and an interactive map appears highlighting zones in red that are danger zones for certain diseases based on certain data

---

Actors:

- User

Preconditions: User is on the “Create” page
Postconditions: User is shown map identifying potential future outbreaks based on trends

9. View Report

R2.1

As a *User* I would like to view a single report
starts when I click on the “View Report” button
ends when Page has loaded

Interactions:

- User selects a report and clicks the “View Report” button
- System redirects to a page containing additional information about that report, which includes a link to the source of the report

---

Actors:

- User

Preconditions: None
Postconditions: User is viewing information of selected Report

9. Extend Search

R5

As a *User*, after the events of UC3.*, I would like to collect relevant reports based on the current reports and NLP entities displayed on the summary page.
starts when I right click on NLP entites or reports
ends when Click one of multiple options to extend search

Interactions:

- User accesses the website
- System directs them to hompage
- [Optional] User searches by criteria as in UC2
- User clicks on the reports they would like to analyse (or alternatively click the “Select All” button if they wish to see a more complete analysis)
- System adds selected reports to a list in preparation of summarising them
- User clicks “Get Summary”
- System redirects to a “Summary” page containing a list of all reports selected, allowing for further anaylysis on the page
- I right click on NLP entites or reports
- I click one of the "Search by ..." option of the context-menu

---

Actors:

- User

Preconditions: Summary page loaded successfully
Postconditions: All relevant reports (according to the search criteria) is added to the list of reports in the summary page

---

System design and implementation

Software Architecture

Front-end

- The web application was built using the React Javascript framework, as well as ReactStrap + Bootstrap for the visual element of the website
- Heatmap visualisation provided through the use of the Google Maps API
- Natural Language Processing provided through the TextRazor API

Back-End

- The web scraper was built using Python, as well as the Re and BeautifulSoup Python libraries.
- The web API was built using the Python Flask REST-plus framework.
- The machine learning model was built and trained using sklearn, Pillow, numpy, pandas, seaborn libraries.

API Endpoints

Return all reports currently in collection

- `GET` **/reports/**

Filter reports

- `GET` **/reports?{params}**

Fetch a singular report

- `GET` **/reports/{id}**

Deletes a report

- `DELETE` **/reports/{id}**

Updates an existing report with form data

- `POST` **/reports?{params}**

Updates an existing report

- `PUT` **/reports/{id}**

Team Organisation

Thoughout the development of the project, our team continued to work well together - we frequently communicated via Facebook Messanger, referred to the Kanban board for sprint deliverables and other tasks, and utilised Slack to efficiently separate discussions regarding frontend and backend issues.

As expected, some workload overflow for these teams required the assistance of members from the other sub-teams, however this did not hinder our process in either area. If anything, it prompted us to re-evaluate design decisions and our vision for our final product, prioritising features that were more easily implemented in the limited time that we had. This can be most clearly seen in the frontend, as we unanimously agreed that the backend functionality was more important, and if something in the frontend was taking too long (that was not strictly necessary), then we would leave it and come back to it only if we had time to spare afterwards.

As detailed in the Management Information report, the team was split into 2 main sub-teams: backend and frontend development. As we developed our API, we allocated additional responsibilites to different members. For example, Bailey and Nabil undertook the task of integrating the Google Trends and Twitter API with our own, as well as implementing Natural Language Processing to enable the user to create a detailed report on a certain outbreak; Harry was assigned the responsibility of implementing Machine Learning, which helped identify geographic locations in which outbreaks were detected; Estella and Jacob moved from casual debugging (i.e. simply using our website to check that everything performs as expected) to more formal API testing using Postman and writing reports based on said tests. Overall, our team member contributions were as follows:

Name	Role

Name	Role
Bailey Ivancic	• Team leader • Backend developer (Python, Flask, API) • Web crawler and API interface • Natural Language Processing • Google Trends and Twitter API integration • Design Information Report
Nabil Shaikh	• Backend developer (Python, Flask) • Frontend developer (React) • Natural Language Processing • Google Trends and Twitter API integration
Harry Tang	• Backend developer (Python, Flask) • Web crawler and API interface • Machine Learning • Design Information Report
Jacob Wahib	• Frontend developer (React) • Debugger/Tester • Management Information Report • Final Report

Name	Role
Estella Arabi	• Frontend developer (React) • Debugger/Tester • Management Information Report • Final Report

We are happy with the work that each member contributed to the project and would consider it to be an equal distribution of effort from everyone involved.

Project Summary

Overall, we are happy with the final product that was delivered.

In regards to functionality, we believe that we were able to provide a unique solution to the functions that were requested. In developing our solution, we tried to think about what the users of epiWatch would value most in the application, and what would make them likely to pick our solution instead of a solution provided by another team. The precepts that we decided to address were the following: Ease of use and scaling adaptability. In terms of ease-of-use, we believe that the React application we have created is quite intuitive to use, as it provides all the information and functions inside an interface the user will likely be used to.

In saying that, our frontend design can still be improved upon. We believe we were mostly hindered by our lack of React knowledge, and time. Although we were able to visualise our content and features in a simple and easy to digest manner, we were only able to do so on a very basic level with the aid of Bootstrap. In other words, we were not able to exploit the vast number of React libraries and fully realise its capablilites for our project. We believe this came down to a matter of time and priorities, as at the end of the day learning more advanced techniques to utilise in our project would have taken longer than we could afford.

Key Benefits and Achievements

As mentioned previously, we believe our 3 key achievements was the implementation of Machine Learning, Natural Language Processing and integration of the heatmap API.

Machine Learning

One achievement is that any medical professional or field scientist can choose their own image data to feed into the AI with no technical expertise to predict future outbreaks and find new relationships with geographical data and disease outbreaks spreading. Another amazing achievement was applying Spatial Data Analysis to predict outbreak clusters around Singapore on Dengue with 75% accuracy using ArcGIS image data and other non-image features with a Random Forest Classifer model.

Natural Language Processing

Integration of the NLP API called Textrazor meant meaningful, automatic interpretation of report data. Upon loading of the summary page, the component would send all description text of every article selected on the previous page to the API to be processed. What was received was a JSON file of entity names and a list of categories each entity within the text belonged to. Using this data we were able to filter through relevant report based on entities or better defined category names to collect a pool of quality reports/data to further analyse.

Heat Map Google API

Integrating our solution with the Google Map API meant proper and meaningful visualisation of location based data. Every report selected on the summary page would now be a heat zone within the central map. This would be done automatically upon page load. The authorisation of API keys would tedious but simple enough to do. However, after running into problems with compatability we realised that we would need to shape the use of the Google map to our surrounding architure (React). This proved difficult but there were plenty resourced online to help.

Issues

The main issue we encountered was a lack of time, being unable to dedicate the time we would have liked to refine our website due to other academic commitments.The learning hurdles also presented an issue as we had to learn new languages, techniques and experiment with unfamiliar structures while trying to create our front end and back end. This once again magnified our time issues since often we would dedicate a lot of time understanding rather than setting out to complete to the task at hand.

Backend

For web scraping, the description of the report had encoding issues where it would display characters that did not match the UTF-16 character set. For example, characters in the report would display text such as`/X00` in the description.

For the API, issues with the endpoints were becoming more complicated. Due to a lack of proficiency with Flask-Restplus API's I was not able to make the API endpoints in accordance to the specification. After a thorough read through of the documentation, use of namespaces, Classes (manager/mediator design pattern), marshal module and other improvements I was able to completly refractor the code and reduce code significantly.

For machine learning, it was a very difficult task to use the data from the specification to predict future outbreaks. One issue was data quality. There were no features that had a high correlation to disease outbreaks spreading, instead, showing features that measure the impact of the outbreak. Thus, the model had a 10% accuracy ***with high variance*** and had a ***lower accuracy rate*** on the training data than the test data. This could be due to the low sample size in the test data. Being able to accurately convert disease cluster locations on an Image to geojson data for the Google Maps API to read was also a difficult task. Another issue was time which was largely spent training the model and debugging.

Frontend

There were a litany of problems with the frontend, mainly boiling down to our lack of proficiency with React. More specifically, they revolved around our desire to make the website look a certain way or perform a certain action without being able to realise it with our language of choice. These issues may not have been necessarily drastic in the grand scheme of our project, but were simple details that would have enhanced the user's experience. For

example, our website allows for reports to be selected and later be used to generate an aggregate report. To indicate that a report had been selected, we wanted to highlight said report. However, no matter what we tried, we could not find a way to do this, only being able to highlight the first selected report (even though the other selected reports were being correctly added to the set). Eventually, through a lot of trial and error and assistance from a member of the backend sub-team, we were able to achieve what we initially wanted, but at the cost of almost 2 days worth of time. Frontend design, then, became three times more time consuming than we orginally thought.

Another main issue was finding a way to convert the responses we recieved from our API and converting them into a format that we could parse through, extracting the data we needed, or simply changing how the responses were given from the backend in the first place. For example, we chose to use **Axios** to integrate our frontend and backend and part of how Axios works is to add query parameters exactly as they are into the url. This meant that for multi-word queries that involed spaces were added as is. However, having spaces in a url goes against convention and Axios would return an error. By default, Axios would replace spaces with '+', and thus we got around this error by simply changing all the spaces in our parameters through a function in the backend to '+' as well (and vice versa).

Some of our most vexing issues were present in the integration of an additional APIs into our website to assist in the analysis of data from our report summaries. We experimented with several and continued running into errors that would often seem insurmountable.

What We Would Do Differently

One major point of difference we would commit to would be to impose stricter deadlines for ourselves in terms of API development. As the first few weeks simply revolved around initial documentation and bare-minimum proof of API concept (in other words, starting a simple web scraper), we underestimated the time needed to polish our final product. This is compounded with the new trimester system and work load we have yet to accustom to, which lead to periods of intense "crunch" right before the demonstrations.

Futhermore, we would have changed the responsiblities we assigned to each team member at the beginning. It became more and more apparent through the development timeline that although the frontend members were struggling with React to craft a website aesthetic to their liking, it was by far much less time consuming than the tasks required of the backend members. This is not to say that members from both frontend and backend did not assist each other in any way that they could, however it was almost considered a last resort - the time spent explaining the inner workings of the backend to the members who were not all that directly involved was time that could have been spent more productively elsewhere.

In terms of the project itself, we would have liked to have added more features for report analysis. Specifically, we wished to have been able to integrate an API that generated graphs and infographics based on different data sets, eg Zika outbreaks mapped by time and location. This would have added a different visual aspect to our summary functionality that is easier to digest at a glance. One of the main reasons we chose React was due to the availability of many APIs or libraries/functions that would have helped us implement such a feature, yet we did not exploit this enough. If time was permitted, we also would have liked to better hone our Machine Learning capabilities with larger datasets. We regret that we were not able to implement the Twitter API or the Google Trends API (more so due to discontinuation of Trends and it's compatability with Node.js).

It would have been nice to integrate more libraries like Redux and Node into the backend as well - it certainly put a strain on the readability of our code and hindered our functionality without them. Specifically, not using Node.js

was almost certainly the problem when trying to integrate more and more API's, since Node.js handles cross-origin communication faults. However, this was a thought midway through completing the project and the introduction of new frameworks to learn would of probably been the death of us.