Summary of the benefits and achievements

Performance

Our API managed to scrape reports and articles from CIDRAP website daily at 6:00am so it updates data frequently. However, the scraper in the backend still needs improvement on filtering data and processing text and natural language. Our API responds within a few second time to collect and send out reports and articles.

Our application managed to integrate and use APIs from our own API, other team's and Google News ones. It would take less than one minute up to five minutes to retrieve data and plot graphs for better and more friendly display of data for analysis, depends on how heavy the data source is.

The visualisation of the data that we collected takes quite a few time when the results get large since we looped through the results multiple times to compute the report numbers. To improve the performance, we could apply other algorithms that's more efficient to get the data sets.

The complexity of the algorithm we used to analyse the data and create graphs is $O(n^3)$. As an example we will look at the graph of disease distribution in a location. The search results are retrieved with both our API and the EpiPro team's API. These are combined, then the data is analysed.

Analysing the data required 3 nested for loops - one for iterating through the list of articles, another for the list of reports within the articles, and finally iterating through the list of diseases. While theoretically this seems quite complex, most articles only have 1 (and rarely 2) reports, so complexity is closer to $O(n^2)$.

Speed of development and prototyping

For the API, the design is that the scraper and the API backend share the same database. Thus, we can divide the implementation into three components, API backend (getting the data from database), database and the scraper. Then, members of our team worked on those three components separately and integrated them in the end. The process went fast and smoothly.

For web application, we chose Vue.js and Flask for frontend and backend. Since the page of Vue consists of many components and the dependency between different components are not high, each member worked on different components and then integrated all the components in the end by setting up the router properly. So the speed of development could be really fast using Vue.js if we had known the basic rules of Vue.js before we started. However, using the combo of Vue.js and Flask is not a very good option because there aren't a lot of guidance online about how to set up the config files properly. Since Vue.js is newly developed and Flask is lessly used by web developers compared to django and other frameworks, our team didn't find a good documentation about how to connect the Flask and Vue.js and spent a lot of time on it. We chose Flask because we are more familiar with Flask, but using some other backends like Firebase might be a better choice.

Reusability

For API, since we seperated scraper, database and API backend(Flask) into three different components, our code is highly reusable. If we change the datasource that we are scraping right now to a different website, we can still get the correct result without changing the API backend and database. Also, we can also change the database from MySQLI to another database like PostGres, only thing we need to do is to reinitialize the database, and then we can make it work normally without changing the other two components.

For the web application, we have files like "CidrapAPI.js" to store functions about calling CidrapAPI and we import those files in when we call them. So it would be super easy if we want to change the API that we call for data. We can just make another file about the new API and change the file name we are importing. Also, since Vue.js framework is a single page application and the page consists of different components, it's easy to add new components to the page if we want to add some new functionalities to the website. The reusability of the platform is also good thanks to the framework Vue we are using.

Integration

To integrate our application with the API, we used frontend to collect data and render it dynamically on the templates. We used our API along with EpiPro's and Google News API to search for articles based on the user's inputs on the search page by using javascript functions in the frontend. We used fetch API with user's input in body parameters, and get response from our API in json, and parse and render outputs on templates using Vue.

We used Flask to connect our application frontend with backend, and use Python to store user accounts and subscription key terms into the database which we get from user inputs on the frontend. And for visualisation and user interface, everything is done in the frontend with Vue.js by rendering data dynamically, so users do not need to reload pages or wait for response from the server.

Team organisation

Team Member	Responsibilities
Yue Wu	1. Develop the API 2. Delopy the API to DigitalOcean 3. Build the frontend of the web app 4. Call the Google News API for the web app 5. Book the rooms for meetings 6. Write the reports

Blaise Paradeza	1. Develop the API 2. Build the frontend of the web app 3. Call our API and other teams API for the web app 4. Write the reports
Minh Thien Nhat Nguyen	1. Build the scraper for the API 2. The login/logout function of the web app 3. Saving the articles feature of the web app 4. Write the reports
Wenjie Zhao	1. Set up the database of the API 2. Set up the database of the web app 3. Using the graphs to show data for the web app 4. Find the frontend templates 5. Write the reports

Appraisal

Major achievements in project

We managed to scrape the datasource constantly using multithreading and keep updating the database to make the software more reliable and useful for the users to analyse the disease outbreaks. By combining the results we get from our API, other teams' APIs and Google News API, we strengthened the reliability even more.

By displaying the data in the format of cards and graphs, we managed to make our website user-friendly. By visualising the data into different types of graphs, we clearly displayed the trends of the development or the distribution of the diseases according to the users' inputs.

Moreover, we provided the log-in system for the users to create their personal accounts so that they can get access to our save and subscribe features. Our users can save the articles and news that contain the information they want to keep to their profile page so that they can check those out easily whenever they need to. The subscribing system provides the users the chance to get the lastest news and articles about the diseases they want to keep tracking so that they don't miss out the potential disease outbreaks that they have been doing research on.

Issues/Problems encountered

In the first stage of the project, we were required to build a scraper to collect data reports and articles from our chosen source CIDRAP, and filter out data based on criteria in the reports such as event type, location, number of affected, disease, etc. However, the source provided was not data friendly, so we needed to implement some language processing feature to filter. There was Natural Language Toolkit module in Python, which is what we used for scraper, but the datasets were too large to be deployed and used. Hence, we manually filtered out data, which did not provide high accuracy.

In the second stage of the project, we were asked to build an analytic platform in three weeks. To do the data visualization, we chose to use the framework Vue.js. However, two people of our group didn't even learn javascript before, not even mention frameworks like Vue and react. We underestimated the complexity of the framework and didn't have enough time to learn it. Thus we struggled a lot to figure out how Vue.js works and how to do the data visualization. We also struggled to connect the frontend Vue.js with the backend Flask we are using. Since Vue.js is pretty new, we didn't find a lot of resources online about how Flask works with Vue.js.

Skills we wish we had

(May) I wish I had known javascript very well before this workshop. Since I haven't done COMP2041(where they teach javascript), I successfully passed all previous courses including COMP1531 and SENG2021 and didn't feel javascript was necessary for those courses. I heard of Javascript before, but since it's not compulsory to learn in previous courses, I never really learned it by myself. And I also wish I had known one framework such as React or AngularJS before. If I had known them before, it would be so much easier for me to learn Vue.js and know how single page application works.
(Jemma) The skill that I wish I had is that I wish I knew Vue or other frontend frameworks before. Learning how the framework works and integrating the template to our website is time consuming. I didn't know how to use tools to test the API and application before either. I wish I knew how to use CircleCI, Locust.io or any other tools for testing so that we can test our project more efficiently so that we could better improve the performance of it.
(Blaise) I wish that I was more familiar with Vue.js and JavaScript. A lot of my time working on the frontend was spent learning Vue.js and understanding the dynamics of Vue components.
(Eric) I wish I had learned and got more experience with using Python, especially for processing data (natural language processing). If we had managed to get more data earlier, we could have save time and implement more features to support users of our application, such as notifications, better detailed graphs and charts for comparison, maps, etc.

How would we do it differently

For our API and data source, we could have learned and utilised the Natural Language Toolkit (NLTK) from Python to process text for more accurate reports and articles, which would improve reliability of our API and database. Also, by applying text processing, we could have extended our application to notify users with new articles and news that are of interest to users. We would first try to integrate NLTK module with our API first to try out and test for its use. Once we confirm its application and usability, we can apply it for our final application. From that, we can process data better and provide more user-friendly format to view, analyse and compare, such as graphs with number of affected along with number of reports, charts, spreadsheets, maps, etc.

When generating data, we could use a dictionary with diseases and number of reports and a dictionary with dates and number of reports to improve the performance instead of using multiple loops doing inefficient search for the reports from the results.

In terms of the frontend of our application, we should have started looking for templates and learning the Vue framework at the same time when we're developing the API. Not all of us were working on the API and we could have organised our tasks more efficiently. Integrating the Vue template to our application took a lot of work and time, especially when we were not familiar with the Vue framework. Therefore, we should have started earlier so that we have more time to improve the UI and develop more features.