SENG3011 20T1

Final Report

Medics

Abanob Tawfik	z5075490
Kevin Luxa	z5074984
Lucas Pok	z5122535
Rason Chia	z5084566

Key Benefits/Achievements

Here are a list of the key benefits and achievements of our project:

- We successfully extracted outbreak data from articles of varying formats on the CDC website. Due to the flexibility of our scraper it is likely that it could also be used to scrape future articles.
- We stored outbreak articles obtained by our web scraper in a standard format in a MongoDB database. MongoDB was very appropriate given the JSON representation of the article and report objects, and made it very easy to store and retrieve articles.
- We implemented a REST API and deployed it on Azure. This enabled other teams to use our API and retrieve articles that we had scraped from the CDC website. The API properly handled invalid inputs.
- We thoroughly documented our REST API using Swagger UI, with detailed and accurate descriptions of the input parameters and the output format. This made our API much more usable as users of our API would know exactly how to call our API and what result to expect. We also allowed users to try out API calls from the documentation itself, which meant that users could test our API without writing any code.
- We aggregated outbreak data from multiple data sources (CDC, WHO, Flu Tracker, and ProMed) using different APIs. This meant that users searching for news about outbreaks would not need to visit multiple sources to get their information, as our application put all the information in one place. Our implementation also cleverly freed the front-end components from needing to care about the subtleties of the different APIs (such as their input parameters and output formats), making our code more maintainable.
- We implemented a simple, intuitive and interactive map with search and filter functionality. This made our application very user-friendly, for both experts and non-experts alike, and allowed users to easily perform and refine their searches, and learn about outbreaks that are occurring around the world.
- We converted the articles that we retrieved from the various APIs into small previews that could be viewed by clicking on links on the map. This enabled users to view a summary of the articles without needing to scan through large amounts of information.
- We provided an SEIR model to allow users to view and predict the progression of an outbreak. The parameters of the model could be adjusted, which enabled users (especially epidemiological experts) to experiment with the model and make predictions.
- We implemented our front-end in AngularJS and followed AngularJS standards, which made our application more responsive and maintainable, and enabled us to provide a richer user experience.

Responsibilities

The table below lists the responsibilities of each of our team members over the full duration of the group project.

Team Member	Responsibilities
Abanob Tawfik	● Back-end setup ● Main web scraper for outbreak articles ● Azure hosting and CI/CD setup ● Front-end map component ● Front-end search functionality ● Front-end filter functionality ● Front-end services, including conversion from location to latitude and longitude ● API for storing the results of location APIs ● Report writing
Kevin Luxa	● MongoDB setup ● Back-end API implementation ● API testing script ● Swagger documentation detailed descriptions ● Back-end and front-end date parsing and conversion utilities ● Article retrieval from different APIs and conversion to standard form ● Front-end UI work for the map page, including the appearance of the map, map markers, info windows, article modals and list view ● Report writing ● Proofreading
Lucas Pok	● Main web scraper for outbreak articles ● Web scraper for coronavirus RSS feed ● Implementing the location mapper service in the back-end which extracted locations from an article ● Swagger documentation customisation ● Front-end UI work for the map page, including the search component and retrieving user input ● Implementing the SEIR model page in the front-end, including the additional API endpoint for fetching case counts ● Report writing
Rason Chia	● Group Meeting Organiser ● Swagger Documentation Swashbuckle Implementation ● Front-end Search Setup ● API Test Script Setup ● Deliverable 3 presentation slides ● Report Setup ● User Stories ● Report Writing

Final Reflection

Overall, we believe that the project went well. Our team worked well together throughout the term, and we managed to achieve quite a lot in the time that was available. We also learned a lot about disease epidemiology as well as various aspects of software development, such as web scraping, creating a REST API, writing Swagger documentation, and using the AngularJS framework.

Major Achievements

Many of our key achievements are listed in the "Key Benefits/Achievements" section above, but here are some of the most important achievements:

- We implemented a web scraper to extract information from outbreak articles. This was one of the most challenging parts of the project as none of us had experience with web scraping before, and it was much more difficult than we had anticipated.

- We achieved a well designed, reliable and efficient REST API endpoint hosted on Microsoft Azure. The usage of our API is extensively documented in Swagger, which allows users to try out the API.

- We built a user-friendly, simple and intuitive front-end platform using AngularJS and Bootstrap to be used by Epiwatch researchers and the general public to learn more about outbreaks around the world. Our front-end platform consolidates outbreak articles from multiple sources and displays them on an interactive map.

- We incorporated an SEIR model in our platform to allow researchers to experiment with different parameters and make predictions about current outbreaks.

Issues Encountered

We encountered many issues throughout the project.

- Web scraper

  The web scraper was very difficult to implement as it had to deal with the irregularities of the CDC website. Many articles on the website did not have a consistent layout, and this made it difficult to extract locations, syndromes and even date ranges. There are more details on the challenges of web scraping and how we solved these various issues in Section 2 of the Design Details report.

- Swagger documentation

  The main advantage of the Swashbuckle package is that it automatically generated Swagger documentation for us. However, the automatic generation made things frustrating at times, as we sometimes wanted to override some of the documentation. Swashbuckle was also not extensively documented.

- Using free API services

  One huge issue that arose when developing our map page was the hourly request limit on the OpenStreetMap and GeoNames APIs. The limit meant that we had to sometimes wait an hour before we were able to continue developing the map. Eventually, we devised a solution which involved storing the results that we received from these APIs into our own database for future use, but this required us to create a new API for storing and retrieving these results and ended up being quite time consuming.

- COVID-19

  The COVID-19 situation greatly impacted our team management. Before Week 6, we were still able to meet up in person, and during our meetings we were able to assign tasks and plan our implementation to a much greater capacity. This is because in in-person meetings, we were more focused, as there were no distractions, and we could communicate more effectively, as ‘messages’ were heard instantly and we were able to explain our ideas using diagrams. After we could no longer meet in person, communication was slower and motivation was reduced. We ended up scheduling fewer meetings after Week 6 than we would have if we were still able to meet in person.

- Low communication in some periods

  While we did communicate a lot, there were still a few long stretches of time (5 days or longer) over which no communication occurred. Even if we were busy working on other courses, it would have been helpful to keep in touch. More constant communication would have also helped to lift motivation levels.

In Retrospect

Having now completed the project, there are many skills that we wish we had before the workshop:

- **Experience with deploying a web service** - if we had more experience with deploying a web service, we would have been able to set up hosting on Azure more quickly. We could have also been made aware of better alternatives.

- **Advanced knowledge of Google Maps API** - the Google Maps API ended up being quite tricky to use. Having someone on the team with more knowledge on how to use the API would have been very valuable.

- **More experience with front-end frameworks** - some of us were not very familiar with AngularJS, and had to learn it on the fly

- **More Bootstrap and CSS knowledge** - as most of us are not front-end enthusiasts, front-end development during this project often consisted of repeatedly modifying small bits of code and reloading the application until it appeared the way we wanted it to. More knowledge on Bootstrap and CSS would have saved a lot of time.

- **Experience with extracting information from the web** - web scraping was the most difficult part of the project. Having someone on the team with natural language processing or web scraping skills would have helped us to develop the web scraper more efficiently.

There are also a number of things we would have done differently:

- **Node.js vs C#** - We could have considered used Node.js for our back-end instead of C#. This may have simplified development as we would be using the same language for both the front-end and back-end.

- **React vs AngularJS** - For our project, we chose to use AngularJS over React as more of us had experience with AngularJS. However, if some of us had experience with React it would have been worth trying out.

- **Google Maps API costs -** Currently, our interactive map calls the Google Maps API multiple times on a single refresh. This is inefficient and costly due to Google charging a fixed cost per request. We would attempt to design a better solution to minimise the number of Google Maps API calls to increase efficiency and reduce ongoing costs.

- **Web scraper** - Currently our scraper is less advanced and more hacky than one created with a natural language processor and cannot handle every single case. It would be more desirable for our scraper to rely only on the main text segment of the article.

- **Task management** - At the beginning of the project we created a Trello to manage tasks. However, after Week 6 we stopped using it as it was too much trouble to maintain.