

SENG3011

Final Report

Team: git push --force

Members:

Austin Vuong, Elliott Yu, Cuthbert Zhong, Alvin Cheng, Jeffrey Wu

Overview

The purpose of this report is to discuss the details of the design and implementation of the Outbreaks API created by the team based on the GlobalIncidentMap data source and SafeTravels, a web app focussed on providing travellers with the essential information they need before and after travelling in a post-COVID world.

This report will detail the implemented use cases and requirements of the Outbreaks API and SafeTravels in addition to the design of the systems and its implementation. Further, the report will detail the team's responsibilities and organisation throughout the project as well as a reflection on the project's overall progress.

Use Cases/Requirements

API

Design and Analysis

In preparation for the implementation of the API, an analysis of the data source was conducted to determine the requirements that were essential for the API. This was achieved by observing the structure of the data provided by the data source and understanding how this data can be subsequently scraped and organised within our database.

Data Source Analysis

Each entry provided within the “Outbreaks Global Incident Map” data source is grouped within the respective tables that relate to the disease they are reporting on.

The entries themselves provide details including:

- date/time
- country
- city
- description

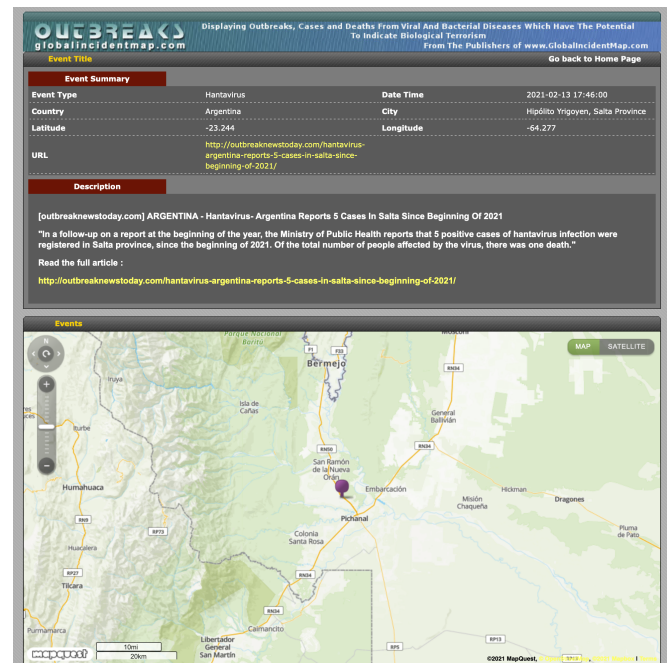
However, each entry is paired with a “details” link that opens a new page that provides the above information that is shown in the table as well as further information including coordinates, a url to the related article and an extended description of the entry.

From this, we can determine that the relevant data can be sourced from both the entry on the “Outbreaks Global Incident Map” data source as well as the article that is referenced in each entry. This will form the data required by the specification of which will be stored within our database and served to the user by the API when queried. The specific sources and form of this data will be discussed in a later section.



DATE/TIME	DETAIL	COUNTRY	CITY	
2021-02-13 17:46:00	Detail	Argentina	Hipólito Yrigoyen, Salta Province	ARGENTINA - Hantavirus- Argentina Reports 5 Cases In Salta Since Beginning Of 2021
2021-02-13 17:45:00	Detail	Argentina	General Mosconi, Salta Province	ARGENTINA - Hantavirus- Argentina Reports 5 Cases In Salta Since Beginning Of 2021
2021-02-13 17:42:00	Detail	Argentina	Colonia Santa Rosa, Salta Province	ARGENTINA - Hantavirus- Argentina Reports 5 Cases In Salta Since Beginning Of 2021
2021-02-13 16:49:00	Detail	Argentina	Ordán, Salta Province	ARGENTINA - Hantavirus- Argentina Reports 5 Cases In Salta Since Beginning Of 2021

Source: <http://outbreaks.globalincidentmap.com>



OUTBREAKS globalincidentmap.com
Displaying Outbreaks, Cases and Deaths From Viral And Bacterial Diseases Which Have The Potential To Indicate Biological Terrorism
From The Publishers of www.GlobalIncidentMap.com
Go back to Home Page

Event Title

Event Type	Hantavirus	Date Time	2021-02-13 17:46:00
Country	Argentina	City	Hipólito Yrigoyen, Salta Province
Latitude	-23.244	Longitude	-64.277
URL	http://outbreaknewstoday.com/hantavirus-argentina-reports-5-cases-in-salta-since-beginning-of-2021/		

Description

[outbreaknewstoday.com] ARGENTINA - Hantavirus- Argentina Reports 5 Cases In Salta Since Beginning Of 2021

"In a follow-up on a report at the beginning of the year, the Ministry of Public Health reports that 5 positive cases of hantavirus infection were registered in Salta province, since the beginning of 2021. Of the total number of people affected by the virus, there was one death."

Read the full article : <http://outbreaknewstoday.com/hantavirus-argentina-reports-5-cases-in-salta-since-beginning-of-2021/>

Event

Map showing the location of the outbreak in Salta Province, Argentina. The map includes labels for various locations such as Hipólito Yrigoyen, General Mosconi, Colonia Santa Rosa, and Ordán.

Source: <http://outbreaks.globalincidentmap.com/eventdetail.php?ID=37805>

API/Parameter Design

From analysis of the data source, we can determine the relevant parameters that will be required to utilise our API. The parameters are discussed below.

The three required parameters as outlined in project specification:

- period of interest/time (start_date, end_date) (*path*)
 - This will consist of start and end date conforming to the format required:
 - (yyyy-MM-ddTHH:mm:ss)
 - The API will expect these dates to be passed in as strings. Therefore, if the API is programmatically called, any date objects that may be used to represent this must be stringified.
 - If certain components of the date are not provided by the user, this should be handled in the front-end for error checking before being passed to the API
 - Any errors that are passed into the API will result in the API returning a 400 Bad Request response
- key_terms (*path*)
 - Key terms provided to the API will be accepted as a comma-separated string of characters.
 - An empty input for this parameter will not result in a failed response. All relevant results of the search based on other parameters will be returned instead.
 - For this parameter, we can expect to determine relevant results by identifying whether these key terms appear in the description of the relevant reports/entries.
 - To narrow our search, the time period and location filters based on the given parameters of the call will be applied first then the key terms search.
- location (*path*)
 - A specific location can form a parameter where the user calling the API intends to only receive reports relating to that specific location
 - As stated, this will narrow the search of the relevant reports to be returned to the endpoint.

Additional parameters:

- disease (*path*)
 - A specific disease can form a parameter where the user calling the API intends to only receive reports relating to a single disease.
 - This will allow for the search to be narrowed significantly.
- syndrome (*path*)
 - A specific syndrome can form a parameter where the user calling the API intends to only receive reports relating to a single syndrome.
 - This will allow for the search to be narrowed significantly.

API Endpoints/Requirements

The endpoints that form our API are discussed below. Also, each endpoint will respond to successful queries with a JSON object conforming to the format required in the project specification. Unsuccessful queries will be handled accordingly with their respective HTTP response codes.

N.B The design of the API and its endpoints were inspired by a sample Swagger doc (<https://petstore.swagger.io/#/>)

Reports Search Endpoint

This endpoint will consist of multiple versions where the user is able to search based on the filters that they choose to provide, which can involve one or multiple parameters (at least one)

- **GET**
reports/?start_date={start_date}&end_date={end_date}&location={location}&terms={terms}
 - eg. start_date = "2020-10-01T08:45:10"
 - eg. end_date = "2020-11-01T19:37:12"
 - eg. location = "Sydney"
 - eg. terms = "Coronavirus,Flu"
- **GET** *reports/?start_date={start_date}&end_date={end_date}*
 - For this endpoint, the response will consist of results filtered only by the provided date range. Both start date and end date must be included, and start date must be before the end date.
 - eg. start_date = "2020-10-01T08:45:10"
 - eg. end_date = "2020-11-01T19:37:12"
- **GET** *reports/?location={location}*
 - For this endpoint, the response will consist of results filtered only by the provided location.
 - eg. location = "Sydney"
- **GET** *reports/?terms={terms}*
 - For this endpoint, the response will consist of results filtered only by the provided terms.
 - eg. terms = "Coronavirus,Flu"

Disease-Based Search Endpoint

This endpoint will be based on one parameter where a specific disease is queried for, returning all the reports for that particular disease

- **GET** *reports/disease?disease={disease}*
 - Only the specified disease will be used as a filter to form the response
 - eg. disease = "Coronavirus"

Syndrome-Based Search Endpoint

This endpoint will be based on one parameter where a specific syndrome is queried for, returning all the reports for that particular syndrome

- *GET reports/syndrome?syndrome={syndrome}*
 - Only the specified syndrome will be used as a filter to form the response
 - eg. syndrome = "Meningitis"

SafeTravels - Web Application

Design and Purpose

Inspired by the global nature of the datasource utilised in Phase 1 of the project (Global Incident Map), the team decided on focussing our efforts on creating a platform that would aid travellers in avoiding infections from disease outbreaks. Through design thinking and further discussions with our mentor, the team had a priority in determining what the end value of the platform should be and how it would be different to existing systems. As such, the team decided to frame the aim of the eventual platform to create in the lens of users who have or will be affected by the current, ongoing pandemic.

With international travel being one of the main industries being affected by the pandemic, the team saw value in creating a platform that would aid in its resurgence after the pandemic. Inevitably, through the experience of the economic, social and health impact of the pandemic, future travellers will be more wary of disease outbreaks in foreign countries and thus want to be more proactive and careful in making decisions related to travel going into the future. This can only be achieved and encouraged through the provision of relevant information to many travellers. With much of this crucial information being reported in long-form formats, such as articles and case reports, this information, to a degree, can be inaccessible in terms of complexity and user friendliness.

As such, the overall aim was to implement a platform that would collate all essential information for travellers in regards to global outbreaks into summarised, readable and user friendly formats. This would bring important details such as report numbers, frequently reported locations, types of diseases to the forefront and out of lengthy reports. Further, the team saw value in further developing the use case of the platform, dividing its use into pre-travel and post-travel use cases to ensure that users will always receive information that is relevant to them at any stage of their travelling.

Use Cases

Pre-Travel

A user of SafeTravels will be able to plan out their trip overseas through the Pre-Travel functionality of the platform. Through providing specific countries that they intend on travelling to, the user will be able to receive contextual information that informs them on medical travel advice, report numbers, important headlines and vaccines to consider. This use case is ideal for travellers who want more confidence in their travelling decisions without the need to read long-form reports to gather the essential information to make these decisions.

Post-Travel

A user of SafeTravels will also be able to use the platform to receive information about their trip after they have travelled. Through providing the countries they travelled to as well as the dates of their travel, they can access summarised data that is contextual to the countries and the times they were in those countries. This allows the user to focus on relevant information 'at a glance' rather than requiring them to determine the relevancy of multiple articles. In addition, a data visualisation feature aids users in determining the likelihood of infection and degree of severity of certain outbreaks in certain locations of which they can relate back to their travel to become aware of whether they may have been infected by an outbreak during their time overseas.

Requirements

Pre-Travel

- As a user, I should be able to view current disease outbreaks for specific countries before I travel, so that I can confidently make decisions about my travel
 - Acceptance Criteria:
 - Disease outbreak information based on country should be summarised into readable formats for the user
 - Disease name
 - Report count
 - Most frequent location
 - Articles
- As a user, I should be able to see travel advice for specific countries before I travel, so that I know what to be prepared for and look out for during my travel
 - Acceptance Criteria:
 - Medical focussed travel advice based on country is accessible to the user
- As a user, I should be able to see vaccination advice for specific countries before I travel, so that I am aware of the essential vaccinations for me to receive to keep safe while travelling
 - Acceptance Criteria:
 - Vaccinations advice based on country is accessible to the user

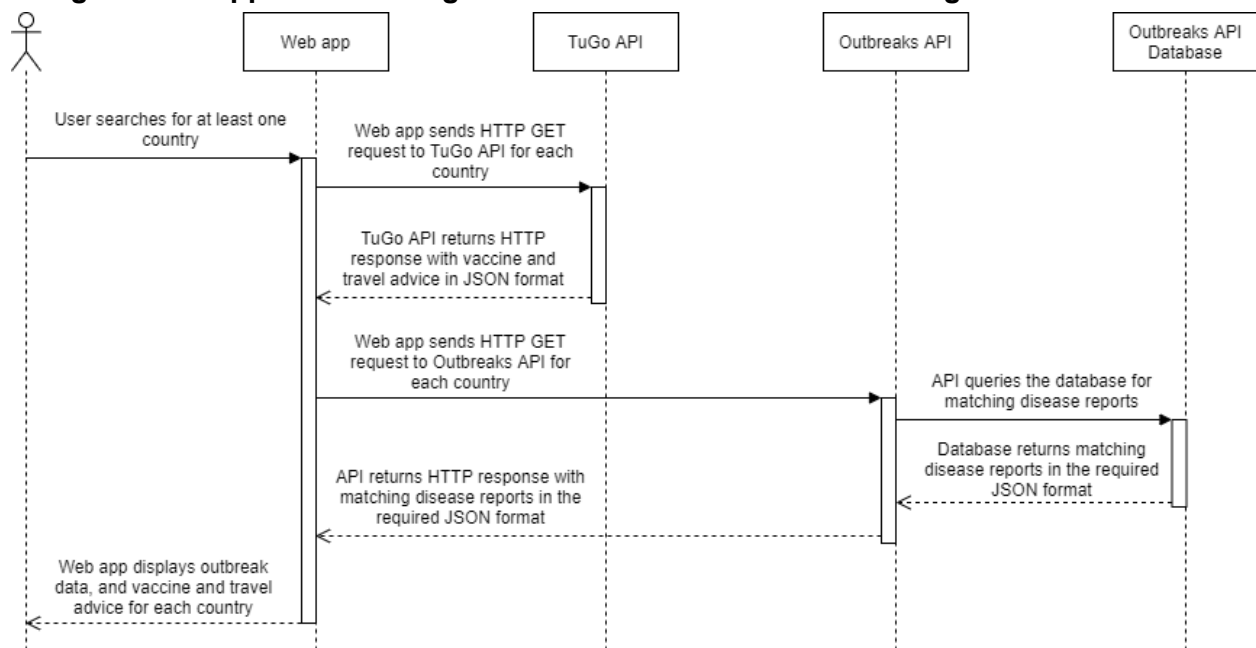
Post-Travel

- As a user, I should be able to see what outbreaks happened in a country while I was there, so that I know what symptoms to monitor for
 - Acceptance Criteria:
 - Countries travelled to can be inputted to retrieve relevant information
 - Date range of travel can be inputted to narrow search to outbreaks relevant to the trip timeline
 - Summary of reports, symptoms and diseases accessible to user
- As a user I should be able to see a visualisation of the reports of various disease outbreaks so that I can easily determine the likelihood of infection based on the locations I travelled to
 - Acceptance Criteria:
 - Charts that are divided by disease to show counts of reports of the disease at particular locations

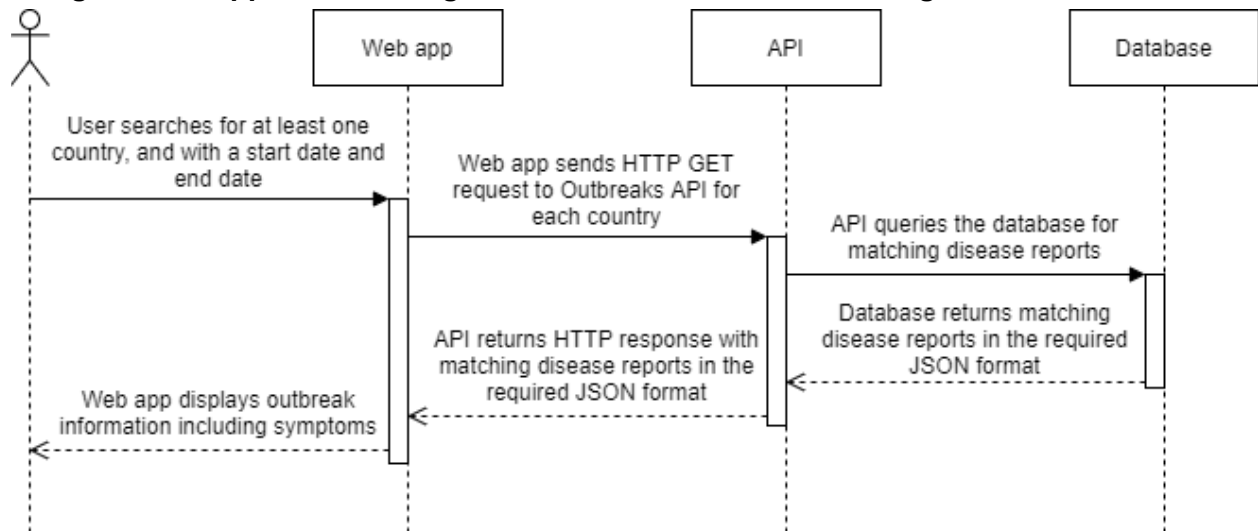
System Design and Implementation

Interaction Design

Using the web app for searching for travel advice BEFORE travelling



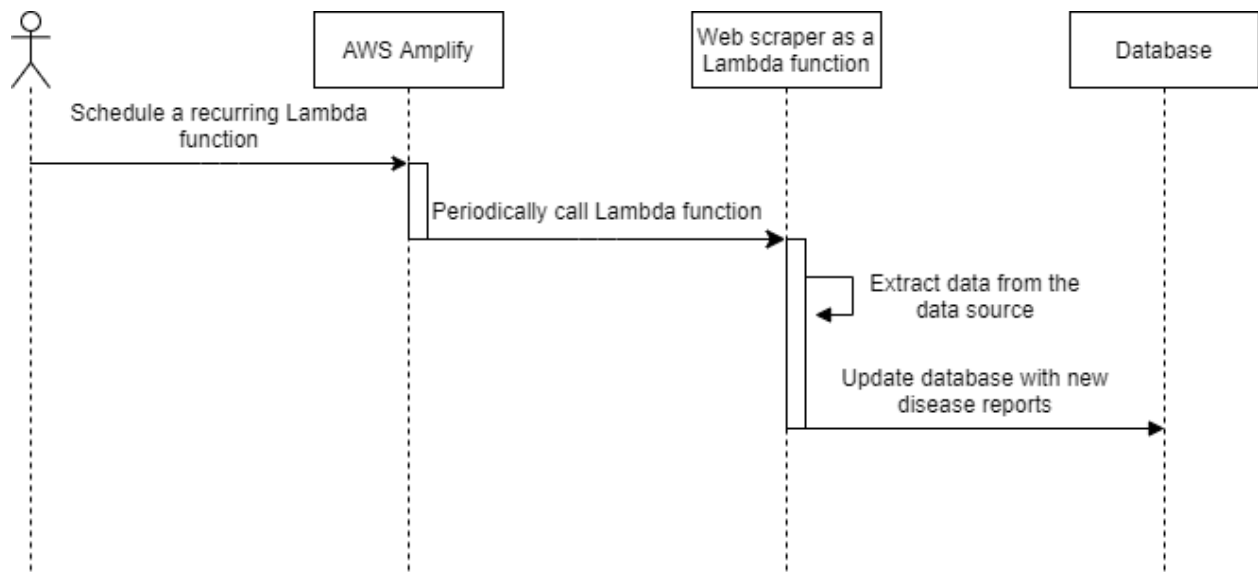
Using the web app for searching for travel advice AFTER travelling



A key difference in the diagrams for a user searching for travel advice either before or after travelling is to do with the different information displayed. For pre-travel advice, we incorporated the TuGo API to provide us with country-specific advice, which was not needed for post-travel advice. In contrast, post-travel advice would display disease symptoms information.

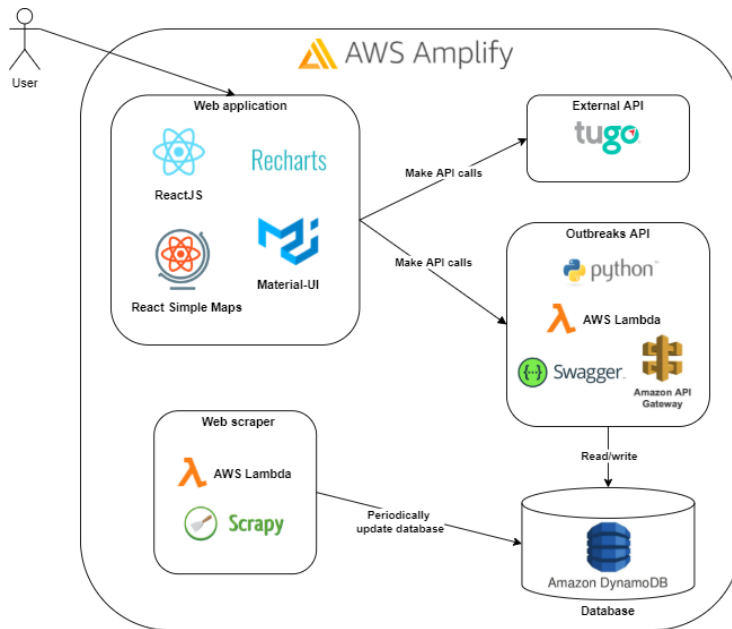
Web scraper

In order to extract data from the intended data source, a web scraper is required. This web scraper will also be deployed on AWS Amplify as an AWS Lambda function which will be periodically triggered to update the database with the latest data from the data source.



Software Architecture

As an overview, the software architecture described by the diagram below consists of the team's Outbreak's API which is hosted on AWS Amplify and the Web application frontend developed in React.



The Outbreak's API itself is the main source of data for the platform. The API is supported by an AWS DynamoDB database to serve data from. An AWS Lambda function which consists of a web scraper developed in Python with the scrapy library periodically updates the disease outbreaks data stored in the database. As such, when the API is queried, a response will be formed based on the data existing within the DynamoDB database. The API is hosted on Amazon's API Gateway with AWS Lambda as well. Documentation of the API is done through Swagger.

As for the web application, the front-end was developed in React allowing for the development of reusable components and efficient development. A key point in our choice to use React was also the abundance of libraries and documentation available due to the framework's popularity. For example, our team was able to leverage the Recharts and React Simple Maps libraries to help us achieve our goals given the tight time frame. Additional APIs utilised in the platform were also implemented in React/Javascript on the front end. This design choice was chosen due to the time constraints of the project. With more time, this logic would have been moved into our existing API to improve performance on the front end and leverage the existing infrastructure that was developed in the early phases of the project instead of adding additional responsibilities to the front end.

API

Web Scraper

Scrapy

Scrapy was chosen to be the main tool to scrape our data source for the team's API implementation. It is a tool specially created for downloading and saving data from the web which makes it more suitable for the project than other frameworks like BeautifulSoup or Selenium. It also outputs the result in JSON format such that it can be efficiently accessed and processed in conjunction with our NoSQL based database (DynamoDB).

Due to the nature of our data source, it was challenging to collect every piece of data from the articles linked from our data source. The linked articles are from more than 50 different websites, meaning that individual spiders would need to be created to crawl each type of website with a different set of rules. The team has decided that this is unfeasible with the given timeframe for the project. However, it was still possible to collect detailed summaries of those articles via scraping each data entry, not the linked article. The linked articles were still crawled down for collecting some key terms with basic spiders.

Python

Python is an easily extensible and widely supported programming language. It is designed for Rapid Application Deployment and as a glue language to connect existing components together. (*What is Python? An Executive Summary, 2021*). Python's high level syntax also emphasizes program readability and reduces the cost of program maintenance.

Most importantly, Python is the language on which Scrapy is built upon and is supported by a variety of software and infrastructure providers such as AWS, Microsoft Azure and Google Cloud. As such, the compatibility of Python with existing components as well as its wide support across a wide range of software and infrastructure services makes it an obvious choice for a programming language for the backend of our API.

AWS Lambda

As mentioned previously, our programming language and web scraper are written in Python. Due to the requirement for our scraper to be hosted on a serverless design, it is essential that the compute service that we choose is compatible with our technology stack as well as efficient for development.

- Both Google and Amazon provide software to accelerate backend development.
 - However, Google Firebase only offers serverless functions in JavaScript/TypeScript leaving AWS Lambda as the only service that offers Python support with substantial backend acceleration capabilities through AWS Amplify.
- Additionally, Lambda functions can also be run at scheduled times, which allows our web scrapy to periodically fetch data and update a database without the need for an always-online server.

Database

Amazon DynamoDB

As mentioned previously, a NoSQL database is preferred for the storage of data generated from the web scraper. Due to AWS Lambda being the choice for a serverless compute service, DynamoDB is a natural choice for a NoSQL database. As an offering by AWS with tight integration with Amazon's other offerings such as Amplify and Lambda Functions, DynamoDB will greatly improve the speed of development.

API Hosting

The API implementation involved utilising several AWS services with AWS Amplify providing a layer of abstraction and both a UI and CLI for setup. These services include API Gateway for API management, AWS Lambda as the backend for the API, DynamoDB as the database, and Python as our language of choice.

AWS Lambda

Lambda was chosen as our serverless solution instead of running a server, on an EC2 instance for example. There was no need for a server to be run when no calls were being made, removing the need for resources being consumed when no calls were being made to the API, as well as the overhead in server provisioning and management.

Another selling point of Lambda was its UI allowing developers to edit the function's source code and deploying it directly, and also with a testing page to immediately test any changes. This would immensely speed up the development process by removing the need to constantly push local changes to AWS through the CLI.

AWS API Gateway

Following that, API Gateway integrates seamlessly with Amplify, using Lambda functions for its backend while also providing an excellent UI for easy management and testing of the API. It is worth noting that with this choice in infrastructure, the initial setup of the API was as simple as using the 'amplify add api' command in the Amplify CLI then following a few prompts. This was crucial in allowing our team to focus on development rather than setup.

Although the abstraction provided by AWS Amplify in the setup of the API provided an efficient start to the project, the team still had to learn each service (Lambda, API Gateway and DynamoDB) to some extent to be able to implement the API. This turned out to pose a challenge, especially with access permissions across the different services being confusing to set up, with another AWS service called Identity and Access Management (IAM) involved. Luckily, after the first setup with the help of documentation and guides, further Lambda functions and API endpoints followed the same procedure and our team documented this knowledge to share amongst each other, eliminating the need for everyone to learn the process.

Another obstacle the team came across in the development of the API was debugging, especially with so many different parts and services involved. For example, errors could be from syntax in the Python code, access permissions between API Gateway, Lambda and DynamoDB, incorrect use of Boto3 methods (AWS's Python SDK) when querying the database, incorrect method setup in API Gateway or connecting to the wrong Lambda function, and much more. This challenge was especially prominent due to our lack of experience with AWS services as a team, but we were able to overcome this by learning how to navigate the UI/CLI for each service. For example, Lambda functions can be tested locally using the 'amplify mock function <functionName>' command, or on the AWS console online, both of which show output and logs. API Gateway also has similar testing functionality on the online AWS console which shows the response to the user, as well as from its associated Lambda function with all errors included. Moreover, in our weekly meetings as a team, we investigated errors and learnt how to use these services together to both share the knowledge and ensure everyone was on the same page.

Logging

Our team's chosen method of logging details about the API consists of two parts: a JSON snippet that is returned along with the API response to the user, and in AWS CloudWatch. The snippet returned in the response body describes our team's name, time the API was accessed, and the API's data source. This information is tailored towards the end user and provides them with more context. On the other hand, more technical logs that are helpful to us developers are stored in AWS CloudWatch, a monitoring and observability service. The choice to incorporate this service was driven by the fact that AWS Lambda (the backend of our API) supported automatic logging of request details to CloudWatch, while also allowing the use of Python's 'logging' library to easily enrich these logs. In particular, we log the request event (incl. Query parameters, endpoint called, and referrer such as SwaggerHub) and response object (incl. Status code and body) sent to and from the API, useful for debugging. Moreover, Lambda automatically provides the function duration and memory used, useful for insight into resource utilisation.

Web Application

Front-End

React

In development of the front-end of the web application, React was the Javascript framework of choice. This is due to its widespread and established use across many existing and popular web applications as well as the team's familiarity with the framework.

The use of a Javascript framework as opposed to Vanilla Javascript or simple HTML/CSS was due to the greater complexity required for the application where many aspects required the storing of state in order for certain components of the application to function effectively. For example, when a user inputs the countries they intend to travel to, this data state must be carried across to other components that form the next page.

The team took a component-wise approach to front-end development, a popular method amongst React developers. This involved the development of small, reusable functional components that could be effectively transferred between multiple aspects of the web application with minimal changes. This helped streamline the front-end development process which was particularly beneficial given the constrained timeline of the project.

React Packages

React-Simple-Maps

The original intention to utilise Google Maps' API for the map components of the application had to be revised due to Google's newly introduced pricing model. Alternatively, it was decided that a more streamlined approach would be to use the 'react-simple-maps' package to provide an interactive map component to our users. Instead of using the latest version of the package, a previous version was chosen in order to implement a rotatable responsive world globe map. Documentation of each version was available online and it provided clear examples and explanations for efficient implementation.

Recharts

The team decided to use the recharts js package to provide a quick and easy way to build responsive charts with decoupled, reusable React components. It is reliable and lightweight as a dependency and is also responsive with built-in customisable animations. Finally, the documentation provides great sample code and examples for fast implementation.

External APIs

TuGo - Travel Advisory API

The TuGo Travel Advisory API is a free API that provides travel and medical advice for travellers planning to undergo overseas trips to over 225 countries in a JSON format. This API is called using a HTTP request every time a user searches for information for a particular country prior to travelling. Of particular relevance to SafeTravels are the medical and vaccination information that this API provides, which provides a supplementary source of relevant travel and medical advice in addition to our existing API.

Key Benefits Summary

Throughout the overall project, the team focussed on creating a platform and API with the potential user as a prime focus.

Although our target user for the platform was not yet defined throughout Phase 1 of the project, the team, through analysis and discussion of our datasource determined where the API could be extended to further benefit its eventual users. As such, this involved the inclusion of the diseases based and syndrome based API endpoints. This proved beneficial later in the use of our platform for querying information about certain outbreaks by disease or syndrome.

Throughout Phase 2 of the project, the team, through design thinking, focussed on how we could build a platform that would solve a contemporary issue. With the overarching talk of the media about international flights making a return in the near future, the team focussed its efforts on creating a platform that would help users build the confidence to travel internationally once again in a post-pandemic world which would greatly help the struggling travel and tourism industry. This approach was further expanded through analysis of the various reports provided by our datasource. The team recognised that the common user would have to read these lengthy reports in order to attain essential, unfiltered information to help them travel safely with potential disease outbreaks in mind. As such, recognising this problem, the team felt it was necessary to turn this data into much more readable and summarised formats in order to reach a larger spread of users in informing them on how to travel safely. In addition to this, the use cases were further broken down with the addition of contextualising information provided based on user input, that is, whether a user is planning a trip or has already travelled including the specific countries and the dates they have travelled as well.

Team Organisation

Within this project, individuals were assigned various responsibilities in the frontend and backend development of the app with some working on the same aspects and others working individually. A general outline of team responsibilities for the frontend and backend development of our app is as follows.

<u>Name</u>	<u>Responsibilities</u>
Jeffrey Wu	Development of RESTful API: <ul style="list-style-type: none">- Local development utilising Python- Deployment of API on AWS Amplify- AWS DynamoDB configuration Development of Web Application: <ul style="list-style-type: none">- Post-travel page:<ul style="list-style-type: none">- API calls- Parsing API response logic- Front-End components
Elliott Yu	Development of Web Scraper for Data Source: <ul style="list-style-type: none">- Local development utilising Python and the Scrapy Library- Deployment on AWS Lambda for database periodic refresh Development of Web Application: <ul style="list-style-type: none">- Home page:<ul style="list-style-type: none">- Interactive map development
Cuthbert Zhong	Development of Web Scraper for Data Source: <ul style="list-style-type: none">- Local development utilising Python and the Scrapy Library- Deployment on AWS Lambda for database periodic refresh Development of Web Application: <ul style="list-style-type: none">- Pre-travel page:<ul style="list-style-type: none">- Integration of Tugo Travel Advisory Api
Austin Vuong	Development of RESTful API: <ul style="list-style-type: none">- Deployment of API on AWS Amplify Development of Web Application: <ul style="list-style-type: none">- Pre-Travel & Post-travel & Home page page:<ul style="list-style-type: none">- Front-End components- State management Team Coordination and Presentations
Alvin Cheng	Database, Testing and Documentation: <ul style="list-style-type: none">- Database Design and configuration of DynamoDB- API Testing- API Documentation Development of Web Application: <ul style="list-style-type: none">- Post-Travel page:<ul style="list-style-type: none">- Data visualisations through Recharts

Conclusion/Reflection

As a conclusion/reflection, each team member has provided a response to each question as indicated below.

How did the project go in your opinion?

- The project went well overall, and the team was able to complete all deliverables in a timely manner. The organisation in the team in terms of having weekly meetings was very well structured, allowing us to align on each other's expectations and communicate on our ideas and progress more effectively.
- Overall, I think the project went quite well. As a team, we were able to continually complete deliverables to a high standard due to having weekly meetings outside of our tutorial session as well as having lots of correspondence when collaboration was needed.
- Overall, I think the project went well. There were quite a few hesitations and teething problems at the beginning that affected productivity and coordination of the team. However, through having more structured meetings throughout the semester, we were able to be much more productive in our decisions and our work, resulting in a great end product.
- I think the project went very well. We were able to continually meet deadlines and deliverables and whilst there were some hiccups and problems, we were able to communicate and solve them and end up producing a very nice product.

Major Achievements in project

- A major achievement the team accomplished was to do with our web app; going from an idea that was just to collate and display information and satisfy spec requirements without a real purpose or value to users, to an app we could follow a persona's story with, and that gives them a reason to use it.
- We were able to create a functional API and database using AWS Amplify, DynamoDB and Lambda, all new technologies that we were not familiar with. Furthermore, I was able to learn and create documentation and automated testing for our API using Swagger and Postman.
- Some major achievements during this project would be having to implement and deploy API publicly and understanding overall software architecture deeply. Our team had a successful and meaningful weekly meeting which didn't usually happen in my past CSE courses. I also learned a new open-source framework "Scrapy".
- A personal major achievement was developing new project management skills. Throughout the project, I had the opportunity to coordinate the team, help members with challenges they encountered and problem solved a variety of problems. In addition to this, having had limited experience with AWS prior to the project, I was pleased to have extended my experience with it during the project.
- A major achievement in the project is getting exposure to AWS. Prior to this project, I had experience with Google Firebase but not AWS. Another major achievement of this project was the design and development of an API using serverless architectures.

Issues/problems encountered

- One of the more evident issues our team had encountered was the gap between what we had wanted to accomplish and implement for each deliverable, and what we actually had time for. Some examples are the authorization layer for our API as a means to rate limit users trying to mass call our API, and some more extensive features for our web app (such as a design overhaul), both of which we simply just did not have time for.
- Despite our success, we initially struggled with deciding how to actually create and host our API and even when we finally decided on using AWS, it took a lot of trial and error to figure out how to actually use it. Another issue we encountered was that the API and database became increasingly more difficult to use and change rules for with the enormous dataset that we had whilst still following the restrictions that we had using the free AWS plan. Finally, while we did manage to have regular catch ups, we often struggled to complete work outside of them, resulting in decreased productivity during the catch ups and tutorials sessions.
- I have encountered issues with Google Maps API being a paid service, not free like it used to be. Alternatively used built in react package to implement interactive map components for the front-end part of the project.
- At certain points, communication could have been improved. There was small amounts of communication outside of our weekly meetings which caused confusion on progress. However, this was mitigated with a 'stand-up' style start to each weekly meeting (ie. what have you done in the past week?, what are you doing next?, do you have any challenges/blockers?).
- Slight challenge coming up with a feasible idea for the second stage of the project. Team seemed to lack direction in terms of coming to a common consensus regarding the direction and goal of the final app for a week.

What kind of skills do you wish you had before the workshop ?

- Experience or exposure to different frameworks would have been nice going into this workshop, to have a bit more context around what kind of different services we could host our API and web app on. Some experience with front end development would have also been great.
- Before this course, I wish I had knowledge about how to actually set up and use the AWS platform as this would have helped tremendously in the beginning stages of our project. Knowledge about API documentation and testing frameworks such as Swagger and Postman would have also helped.
- I wish I had better front-end skills like JavaScript / React JS so that our webpage doesn't look plain.
- I wish I had more experience with AWS. This would have helped our API development greatly in terms of efficiency and functionality.
- Experience with AWS and PostMan.

Would you do it any differently now ?

- If time permitted, being able to design the web app more extensively before going straight into coding would have been nice to foresee the different design challenges we encountered. For example, the information we displayed was all in a single column requiring a lot of scrolling, resulting in fair amounts of whitespace on the left and right sides of the screen, and perhaps some high fidelity prototypes could have helped avoid needing any design overhauls.
- If I was to do the project again, I would make sure that we wouldn't hesitate as much in the beginning to select a hosting platform for our API like we did for this project, hopefully allowing us to better allocate our time to actually creating and building the API.
- I would have taken a more design-focussed approach to the web application rather than an implementation-focus due to the time constraints. This would involve prototyping extensively so that development would require fewer revisions and result in a more streamlined process as a result.
- Creating a better designed app in terms of user interface and design. The frontend that was created was not the easiest or most intuitive to use and the user experience could have been improved with further work.

Bibliography

- *What is Python? Executive Summary* (2021), <<https://www.python.org/doc/essays/blurb/>>.
- Swagger n.d., *Swagger Petstore*, Swagger, accessed 1 March 2021, <<https://petstore.swagger.io/#/>>.