We Byte- Deliverable 2

Note: Data source is promedmail.org

1.1 API Design Details

Describe final architecture of your API, justify the choice of implementation, challenges addressed and shortcomings, in your Design Details report in Github (or equivalent).

1.1.1 API Summary and Endpoints

Our API uses scraped articles from ProMedMail.org to provide end users with an accessible source of disease report data.

We currently have three endpoints implemented, with a fourth (free-keyword) listed but not currently working due to very high processing time. Our three currently working endpoints are:

/search-keywords

This endpoint allows a user to search for articles based on keywords, location, published date, and allows users to either retrieve the main_text of the article, or a summary. The search-keywords endpoint is used for filtering articles in many ways, and provides the client with articles that are relevant to them.

/latest-articles

This endpoint allows users to search for the n (default 50) latest articles based on date published. This allows clients to display recent information to their users. Once again, we are able to choose between summary and main text in this endpoint.

/article

This endpoint allows users to request a specific article ID. This endpoint is useful in the case a user has the summary of an article, but wants the full main text. Our articles use the same ID's as listed on ProMedMail.org, so this also helps teams who may want to compare results from other API's who have also scraped from ProMed. We allow users to choose between main text and summary, with main text being default.

Each of our endpoints retrieves object(s) of type article, which each contain article metadata, article text or summary and a list of reports which each contain locations, diseases, syndromes and event_dates relevant to the article. We return a list of reports for each article, however there currently can only be one report per article due to limitations in our ability to extract separate cases from an article.

To see more on our endpoints and the specific data they retrieve, please see section 1.3.1 containing a link to our SwaggerHub page.

1.1.2 Web Scraping

We approached scraping as a two stage process.
Stage 1: Scraping the initial data
Stage 2: Keeping the Database up to date with new articles

Both stages required the data to be first collected from ProMed, and then stored in our database. Because of this, we were able to reuse a lot of code, and the main things that had to be added was code to identify which articles had not yet been scraped.

Initial Scraping

For the initial scraping process, it was important to scrape as much data as could to fill out our initial dataset, and make subsequent scraping quick and effective. Due to the amount of data, this process took a long time, and we ended up with over 23,000 articles in the database.

The initial scraping was done in two steps. Once we had confirmed that the data was being scraped correctly and we could process it into reports, we began scraping into text files. The web scraper would go through the list of keywords from the spec and get all article IDs associated with each keyword using the ProMed search page. It would then scrape each of these articles and dump a json with all their details and associated reports into a text file named after the keyword. Scraping all keywords like this took many hours, at approximately 2000 articles per hour. These speeds however, were much faster than our initial speeds due to our incorporation of multiprocessing, allowing the scraping of 10 articles at once.

Once we had all the text files, they were parsed and uploaded to our postgresql database. This process was much quicker than the scraping, especially once we optimised the upload function to operate in batches, rather than uploading each row individually.

Some statistics about our data after the initial scraping:
- 23.9k Articles
- 23.9k Reports
- 248k Locations in Reports
- 48k Diseases in Reports
- 5.1k Syndromes in Reports
- 139k Date/Times in Reports

Subsequent Scheduled Scraping

The second stage occurred after the database was populated with the data from stage 1. This process involved scraping only the new additions to promed and further populating the database with the newly scraped data.

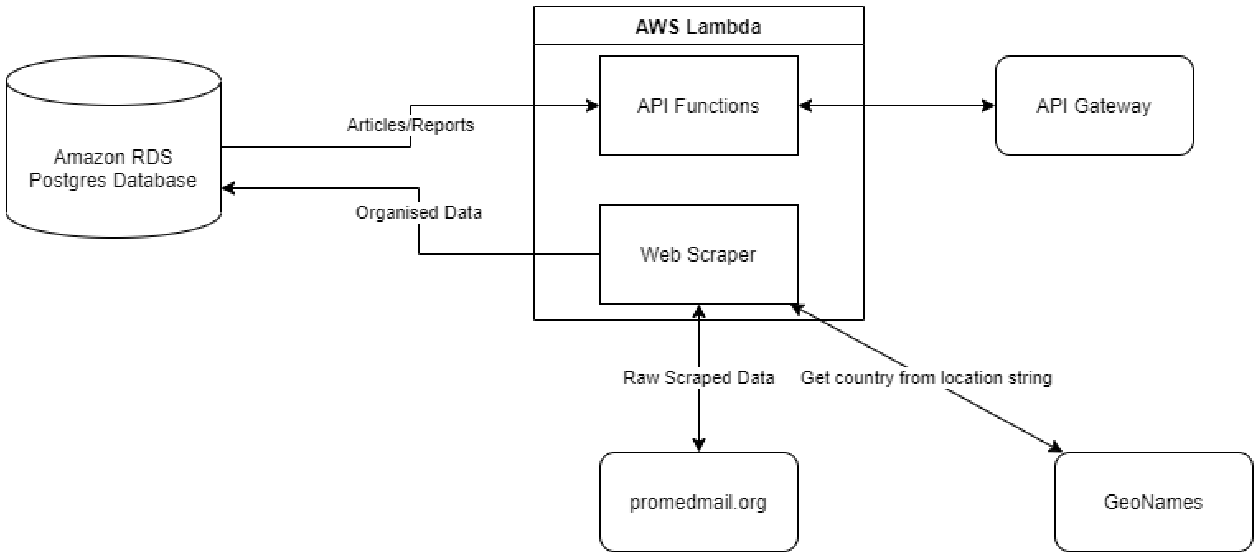
Given AWS Lambda had time constraints that we needed to adhere to, the implementation of the web scraper was subsequently split into the 2 stages.

As a team we observed that promed gained approximately 2-10 new articles a day which meant that the second stage scraper wouldn't need to be as memory or performance intensive as the first stage, adhering to the time limitations set out by AWS (i.e. Lambda functions can only run for a period of 15 minutes).

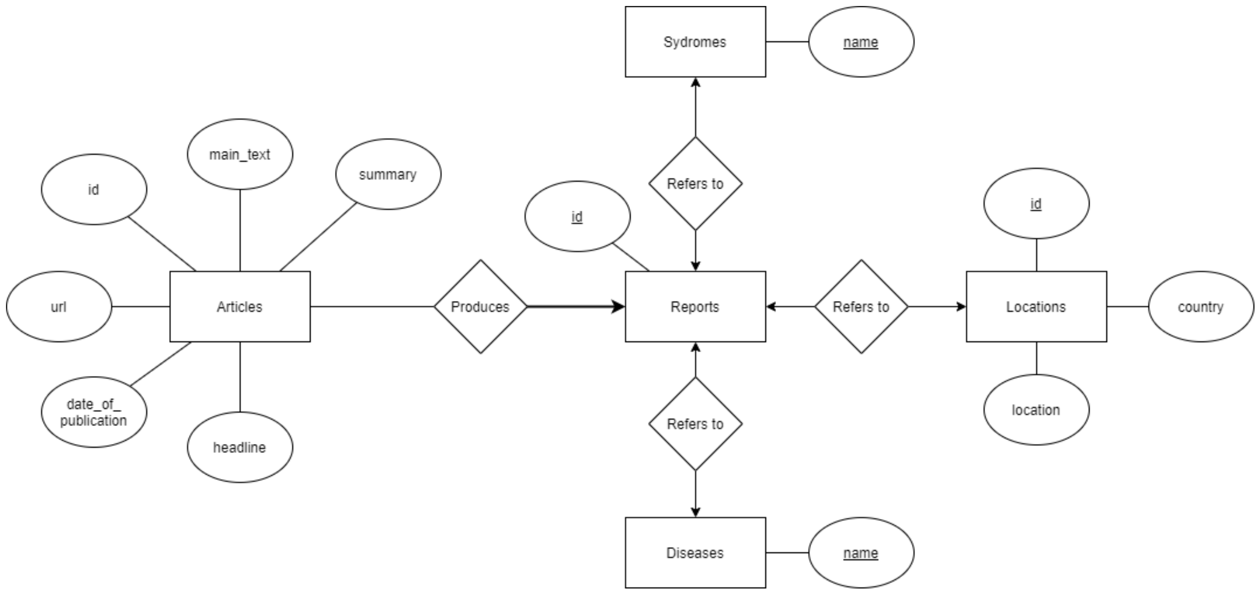
In order to identify the articles which were yet to be scraped, the scraper queried the number of articles existing in the database per keyword and subtracted this value from the total number of articles in the promed posts database. Finally, this value was passed into a promed url to retrieve n number of new articles and then further utilised the same logic in the first stage to extract the relevant information and populate the database.

1.1.3 Architecture Diagram

The following diagram details the final architecture used in the creation of our API, and illustrates the process of extracting data from our data source: "promedmail.org" until it is returned after an API call.



1.1.4 Database ER Diagram



1.1.5 Implementation Justification

Implementation Language: Python

	Python	**PHP**
Team familiarity	5/5	0/5
Web Frameworks	Many web frameworks	Access to mature frameworks
Speed	Interpreted language, slower	Faster than Python
Open Platform	Designed with features/libraries to facilitate data analysis and visualisation	Platform independent
Web Service Libraries	Easy integration with flask and/or django	Large array of libraries available
Industry Relevance	Industry Leader, many deployment options	Open source, easy development
DataBase	Doesn't support database connectivity as much as PHP but RDBMS like PostgreSQL, MySQL, SQlite are still accessible	Able to access more than 20 different databases
Dependency management	Pipenv	Php composer

Readability	Indentation requirements makes it makes it very readable	Highly documented- classic approach
Debugging	Faster	Slow
Usability	Very versatile - able to use it for web development, robotics, science etc	Mainly used for web development

Python had the greatest appeal for developing our API given its flexible nature, ability to integrate seamlessly with many web frameworks, cross-platform design and simple integration into the rest of our chosen web stack. Additionally, all team members were highly familiar with Python, making it the best language to use to ensure our team was easily able to collaborate and develop across multiple platforms. Finally, the abundance of modules and libraries that come alongside Python like multithreading libraries, selenium, BeautifulSoup etc meant it has added versatility and aided when expanding upon each of our API endpoints. Despite the advantages that come along with using PHP and other backend programming languages like access to mature frameworks, speed and community support, the teams' familiarity with Python and simultaneously its simple integration with a multitude of libraries and frameworks meant it was the best suited language for our project. Ultimately after reflecting on the implementation of the API, Python proved to be the right choice and served as the most optimal language for this task.

Platform Provider: Amazon Web Services

There are many cloud platforms to choose from, however, AWS was chosen over the others due to the reasons listed below:

- **Prevalence in industry**: it is by far the most used singular cloud service. With a market share of 33%, with its next biggest competitor Azure at 13%.
- **Large ecosystem**: of over 200 cloud products. Making it easy to deliver more complex microservices.
- **It's free tier:** offers an abundance of database storage and compute time. More than enough for this project
- **Heavily documented:** and has many tutorials, which will help make it easier to learn

Serverless Platform: AWS Lambda & AWS Gateway

We decided to implement our API with a serverless architecture. This meant we would be using AWS Lambda and API Gateway.The reasoning for this is listed below:

- **Easily scalable**: We expect that the usage of our API will have long periods of inactivity then spikes of high activity (such as demo day). Using Lambda provides a lot of horizontal scaling, it defaults at 1000 concurrent requests.
- **Cost effective**: It only gets run when it is triggered, this saves on a lot of expense compared to using a server model in which the server must have long periods of uptime.

The free tier of Lambda includes 1 million requests and 400 000 GB-seconds of compute time a month, as long as each function is limited to a compute time of 5 minutes. This plenty for the scope of this project

- **Modular**: Each Lambda function is designed to serve a singular purpose. This allows us to design a much more decoupled design, making it much more simpler and more manageable. As opposed to an entire EC2 container which would run many different aspects of the API, and therefore a simple bug could compromise the entire API more easily.

Database: AWS RDS- Postgres

	AWS RDS- Postgres	**DynamoDB**
DB Type	Open source relational database systems	Fast and flexible NoSQL database with seamless scalability.
Schema	Required- to adhere to relational database structure	Requires no Schema
Keys	Primary and Foreign Keys (Link to other tables)	Primary and Sorting Keys
Querying & DB manipulation	Simple query syntax & ease of extraction of data	- Query: Only used for primary and sorting keys, requires a primary key to be set to a value - Scan: Expensive and only used for attributes
Community	Large community, and abundance of documentation and help resources	Released in 2012- small community
AWS Free Tier	- 20GB General Purpose Storage - 20GB Database backups and DB snapshots - 750 hrs/month of micro database usage	- 25Gb of Free Storage - Enough to handle up to 200M requests/month
Linking Tables	Use of foreign keys to link multiple tables	Linking of tables is complex and/or not feasible
Ordering/Sorting Tables	ORDER BY & GROUP BY query syntax	Potentially with GSI but complex and expensive method
Ease of Use	Large extraction of data and filtering of table is made simple through large array of queries possible	Steeper learning curve but becomes simpler once you become familiar with the DB
AWS Setup	Complex initial setup of DB instance and interaction of AWS Lambda is similar to any python script	Simple setup and interaction with AWS Lambda is documented and simple

	interacting with a PostgreSQL DB	
JSON	JSON format not applicable with RDMS structure	JSON acceptable

Despite the easy integration of DynamoDB with Lambda its the simple AWS setup, through research and testing the database it became apparent that it was expensive to make queries and extract information. Given the requirement for our API to be efficient and scalable as it would potentially be used by other teams DynamoDB proved not to be the right choice, as detailed in (1.1.3). Ultimately, the team's familiarity with SQL Syntax and the ease of extraction of information from a database meant PostgreSQL was the most suitable choice as detailed in the final architecture (1.1.1).

WebScraping & Libraries

- **Beautiful Soup, Urllib, Requests**

Requests was used to query GeoNames to lookup locations. Urllib was used to query promedmail.org's search functionality to find which article ID's we needed to scrape, and Beautiful Soup was used to parse html documents after they were retrieved by Urllib/Requests/Selenium.

- **Selenium**

Selenium is an open-source automation tool that has easy integration with Python and Beautiful Soup and allows for the automation of browsers which was needed for our web scraper due to article data not being included in the initial get request on proMedMail.org, and only being displayed seconds later by delayed jQuery functions. Rather than trying to interpret the javascript of the page, Selenium and its chromedriver allowed us to simply load the webpage as a human might, and automatically retrieve the data once the page had loaded it for us.

- **NLP- spaCy**

spaCy is a natural language processing library. We used it process the main text of scraped articles, extracting locations that were mentioned, along with dates, diseases and syndromes. Unfortunately, the data from spaCy is not perfect, but we were able to clean the data somewhat by removing duplicates, blacklisting common erroneous locations, and removing dates that could not be interpreted by dateutil's parser. There was some consideration of more advanced use of spacy for case identification, but this was considered too difficult within the given time frame and with our teams current abilities.

- **Psycopg2**

Psycopg2 is a python library for connecting to postgresql databases. This allowed us to upload/retrieve data to/from our postgresql database hosted within Amazon RDS. Pyscopg2 allows easy construction of queries and automatically handles type conversion between python inbuilt types and postgresql types. It also allows batch execution of queries which we utilised to upload our scraped data efficiently, cutting down on round trip times.

Other libraries used in our API, lambda functions and webscraper were Boto3, Json, Requests and DateUtil all which served their own purpose in increasing the functionality of our API, allowing for easy integration across all platforms and/or frameworks and adherence to the specification requirements.

1.1.6 Challenges and Shortcomings

Database: SQL vs NoSQL

Originally we tried to use a DynamoDB database, due to its efficiency and ease of use. However, as we developed our API endpoints, it became more difficult to implement the functionality we wanted with the key-value model used by DynamoDB. Our team's relatively limited knowledge of no-sql compared to sql meant continuing to use Dynamo was going to be very challenging, and we decided to switch to postgresql, which we were more familiar with.

AWS Lambda

While porting our web scraper into AWS Lambda, we hit issues due to the size of the libraries required for our scraper and NLP engine spacy. Lambda imposes a file size limitation of 250mb including source code and all required libraries. We managed to solve this by separating the code that uses the web scraping libraries, and the code that uses spacy to do reports processing into two separate Lambda functions.

GeoNames API

We decided to use GeoNames to lookup country information due to it being free and relatively easy to set up. However, GeoNames has a limit of 20,000 requests per day, and during our initial scrape, we hit that limit quickly meaning much of our location data is missing country information even when it should be clear what country it is from. To mitigate this, we will slowly update the locations table in the database to perform new look-ups of the country where it is currently missing, as our request limit allows. Another option was to switch to Google Places API, however this would have been expensive due to pricing at around $32USD per thousand requests.

AWS API Gateway

Although API Gateway comes with many perks, it proved challenging with its integration with AWS Lambda as it only allowed for 30 seconds for all integration types. After having developed a “free-keywords” whereby a user could pass in a keyword into the url that’s not specified in the specification and the API would scrape the data source live, it became apparent through testing that this development was to no avail as it required a longer period of time to be executed and API Gateway did not support this.

1.2 Testing

Describe the testing processes used in the development of API, referring to the data and scripts included in Phase_1 folder. This should describe your testing environment and/or tools used, and limitations (e.g. things that are not tested). Describe your testing process i.e. how your team conducts testing using the test data (e.g. in which order) and an overview of test cases, testing data and testing results. Describe the output of testing and what actions you took to improve the test results

1.2.1 Testing Environment

Our testing environment involved us running test scripts locally in a virtual environment via pyvenv. We utilised the following Python testing modules to ensure our API's data is robust:

- **Pytest**

Pytest is a Python framework that allows developers to easily write unit tests for their code. We used pytest to test our endpoints, our data and our webscrapper. Pytest also has a pytest-cov library that ensures maximum code coverage.

- **Requests**

Requests is a simple HTTP library for python. It is extremely easy to use and very flexible. We used this library to test our endpoints.

- **Swagger & AWS Tests**

Swagger and AWS "Test" feature was used for manual verification and debugging during development.

1.2.2 Testing Processes (Data and scripts used)

Testing Overview - API Endpoint

To test the API's usability we used the pytest and requests library to make several different forms of requests to the API. We aimed to make simple and complex successful requests, and also tested the boundaries of the API with edge cases and fail cases

Endpoint	**Test Case**	**Test Objective**
search_keyword	*test_search_key*	Test ensured a simple use case of the

	words_basic_200	search_keywords endpoint with one keyword and valid dates would work.
search_keyword	*test_search_keywords_multiple_keywords_200*	Test ensured a complex use case of the search_keywords endpoint with multiple keywords and valid dates would work.
search_keyword	*test_search_keywords_location_200*	Test ensured a complex use case of the search_keywords endpoint with a location, single keyword and valid dates would work.
search_keyword	*test_search_keywords_missing_required_end_date*	Test ensured a use case without a required field; end_date wouldn't work.
search_keyword	*test_search_keywords_missing_required_start_date*	Test ensured a use case without a required field; start_date wouldn't work.
search_keyword	*test_search_keywords_invalid_start_date_400*	Test ensured a use case with an invalid value for start_date wouldn't work (i.e. "banana").
search_keyword	*test_search_keywords_invalid_end_date_400*	Test ensured a use case with an invalid value for end_date wouldn't work (i.e. "banana").
search_keyword	*test_search_keywords_invalid_date_order*	Test ensured a use case with an end_date before a start_date wouldn't work.
article	*test_article_200*	Test ensured a simple use case of the article endpoint would work.
article	*test_article_invalid_id_400*	Test ensured a use case with an invalid value for id wouldn't work (i.e. "banana").
latest_articles	*test_latest_articles_200*	Test ensured a simple use case of the latest_articles endpoint would work.
latest_articles	*test_latest_articles_invalid_n_articles_403*	Test ensured a use case with an invalid value for n_articles wouldn't work (i.e. "banana").

Limitations - API Endpoint

There were minimal limitations in this testing environment, the only possible limitations would stem from overlooking certain edge cases.

Testing Overview - API Data

The API data was tested using a pipeline of 4 shell scripts (one for each checker/check in the flow diagram). Each shell script had an accompanying python script which was used to verify the JSON responses. Each response was checked for a generated report and valid locations. Verified data was then cross examined with the stored data to check that the API requests are being successfully handled and that the data was being stored correctly.

Limitations - API

- The output from the NLP Spacy module is not verified. We are relying that Spacy can extract the report data correctly.
- Currency the testing pipeline has only been set up in unix as is not cross compatible with other OS systems. Therefore testing has to be conducted by specific individuals on demand.
- Load testing hasn't been considered. We are relying that AWS automatically handles that.
- Only archive ID queries are being tested at the moment, other queries such as location, date, keywords or latest articles are not being tested.

Testing Overview - WebScraper

Our Webscraper is divided into 2 sections:

1. One that we originally used to move all the current Promed posts into the database. This is on our local machine as we were concerned about the resources it required and we only needed to run it once.
    a. Tested by connecting to the database on aws and ensuring results were as expected. Libraries used:
        i. boto3 to connect to the database,
        ii. unittest for assertions
        iii. json to load the data returned from the scraper
2. The second one is an updating WebScraper which updates the database once a day (set to automatically occur at 3 a.m) with new promed posts. This is hosted on aws lambda.
    a. Tested on AWS Lambda (see code and results on GitHub TestScripts). Libraries used
        i. boto3 to connect to the database,
        ii. unittest for assertions
        iii. json to load the data returned from the scraper

Limitations - WebScraper

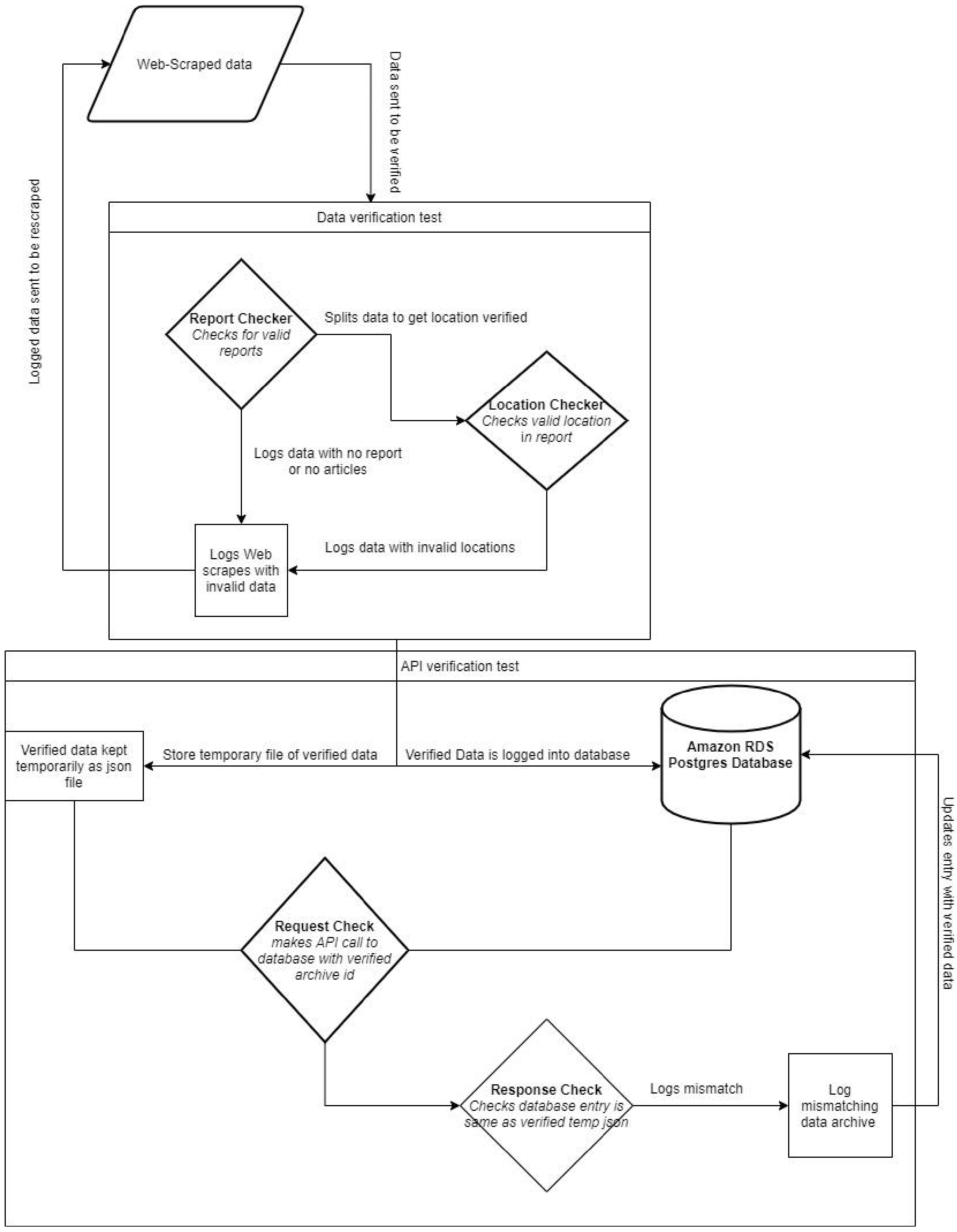
Given our webscraper is automatically scheduled to run every day, the posts that the scraper should retrieve in order to update the database change constantly, thus resulting in difficulties to verify the output was correct. Although at the time of testing a majority of the tests passed, current tests will fail as the new additions to promed will change daily. Finally, testing inconsistencies occurred as despite testing the webscraper locally was successful, it had the limitation of potential environment incompatibility with the AWS version.

1.2.3 Testing Process (How testing was conducted)

The testing process was broken into 2 parts:

- **Data Verification:** Data that has just been scrapped undergoes a verification process whereby the existence of the generated reports are checked (as sometimes they are not created due to multi threading issues in the scrapper) and the locations in those reports are valid as we have a limited amount of request with the GEO encode API. If not, that data is logged and sent to be rescraped.
- **API request verification:**.The verified data is then checked to ensure it has been added to the databases correctly. It also checks that the API is still running and that queries can still be made.

Below is a flow diagram of how it proceeded:


Web-Scraped data
Data sent to be verified
Logged data sent to be rescraped
Data verification test
Report Checker
Checks for valid reports
Splits data to get location verified
Location Checker
Checks valid location in report
Logs data with no report or no articles
Logs data with invalid locations
Logs Web scrapes with invalid data
API verification test
Verified data kept temporarily as json file
Store temporary file of verified data
Verified Data is logged into database
Amazon RDS Postgres Database
Updates entry with verified data
Request Check
makes API call to database with verified archive id
Response Check
Checks database entry is same as verified temp json
Logs mismatch
Log mismatching data archive

1.2.3 Testing Output

Data Verification
The testing demonstrated that there were a lot of archives that weren't being scraped properly. The main text article were never being analysed, thus the reports never being generated. This was linked to an issue with multi threading the web scraper, and is currently being debugged to try and amend this issue. Currently 29/78 scraped diseases have some archives (can vary from 5 to approx 50) that don't generate reports. There are also 15/78 diseases in which no archives are found.

The Location checker also showed how we underestimated how many requests we make to the GEO encode API, already maxing out the free 20 000 monthly requests.

API Request Verification
So far no data mismatching archives have been logged, giving us confidence that the verified data is being inserted correctly, as well as, the API gateway being fully functional for archive queries .

1.3 Documentation

1.3.1 SwaggerHub API Documentation

We created a SwaggerHub page to help users (other teams) learn how to use our API and its various endpoints. We've added documentation, examples and utilised try-it functionality to make it as easy as possible for users to get up and running with our API.

Our SwaggerHub Page

https://app.swaggerhub.com/apis/Seng-We-Byte/We-Byte/1.0.0#/

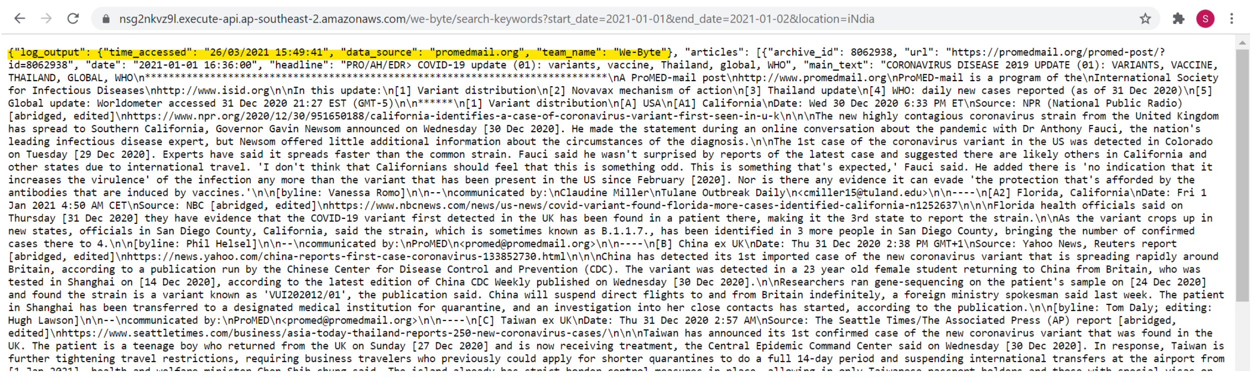
1.3.2 Log Files

For each of the endpoints within our API, a separate log group has been created within AWS CloudWatch to allow for logging of API requests i.e. there are 3 log groups each corresponding to our 3 endpoints- /aws/lambda/latest, /aws/lambda/search-keywords, and /aws/lambda/article_id.

There are 2 different log outputs generated every time an endpoint is accessed: log files stored in the backend i.e. CloudWatch AWS and a log file in the format of a json which is returned to the user every request.

The JSON snippet returned to the user contains the following information, as highlighted in the screenshot below:

- Time accessed
- Data source
- Team name


nsg2nkvz9l.execute-api.ap-southeast-2.amazonaws.com/we-byte/search-keywords?start_date=2021-01-01&end_date=2021-01-02&location=iNdia

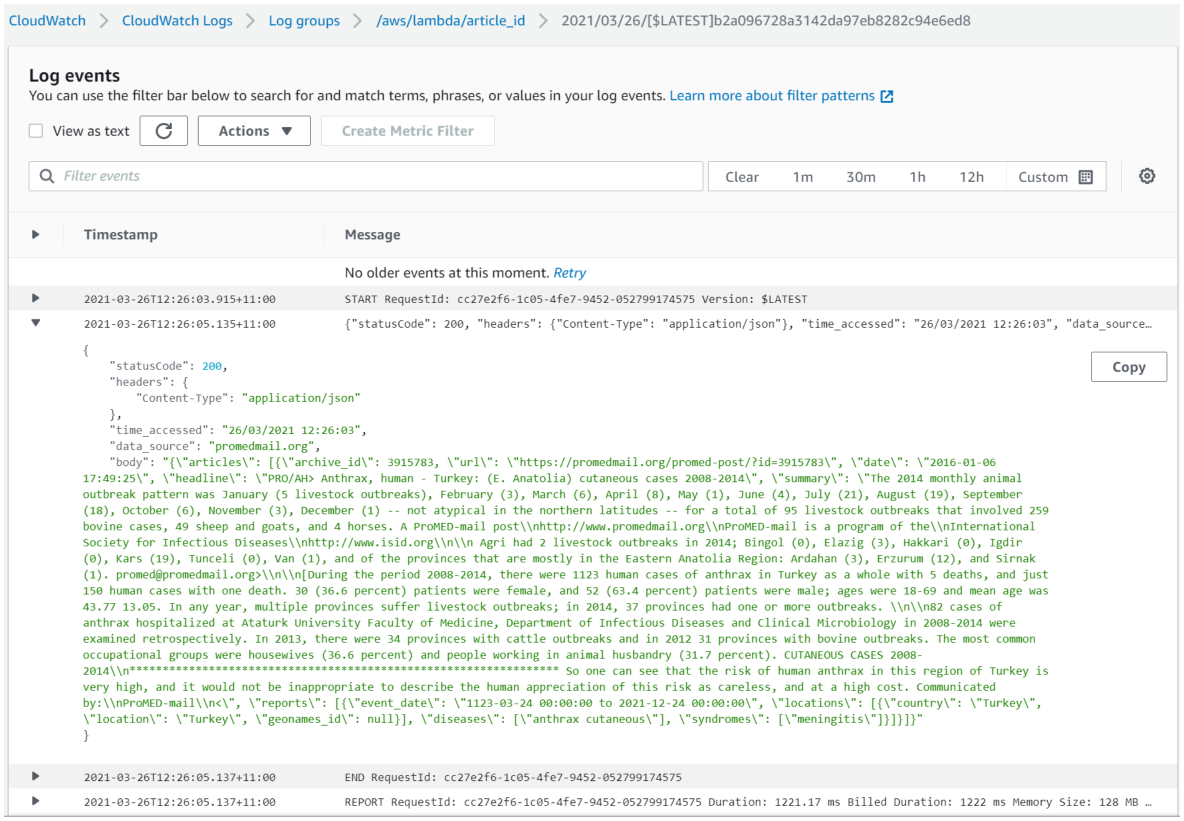
```
{"log_output": {"time_accessed": "26/03/2021 15:49:41", "data_source": "promedmail.org", "team_name": "We-Byte"}, "articles": [{"archive_id": 8062938, "url": "https://promedmail.org/promed-post/?
id=8062938", "date": "2021-01-01 16:36:00", "headline": "PRO/AH/EDR> COVID-19 update (01): variants, vaccine, Thailand, global, WHO", "main_text": "CORONAVIRUS DISEASE 2019 UPDATE (01): VARIANTS, VACCINE,
THAILAND, GLOBAL, WHO\n*****************************************************************************\nA ProMED-mail post\nhttp://www.promedmail.org\nProMED-mail is a program of the\nInternational Society
for Infectious Diseases\nhttp://www.isid.org\n\nIn this update:\n[1] Variant distribution\n[2] Novavax mechanism of action\n[3] Thailand update\n[4] WHO: daily new cases reported (as of 31 Dec 2020)\n[5]
Global update: Worldometer accessed 31 Dec 2020 21:27 EST (GMT-5)\n\n******\n[1] Variant distribution\n[A] USA\n[A1] California\nDate: Wed 30 Dec 2020 6:33 PM ET\nSource: NPR (National Public Radio)
[abridged, edited]\nhttps://www.npr.org/2020/12/30/951650188/california-identifies-a-case-of-coronavirus-variant-first-seen-in-u-k\n\n\nThe new highly contagious coronavirus strain from the United Kingdom
has spread to Southern California, Governor Gavin Newsom announced on Wednesday [30 Dec 2020]. He made the statement during an online conversation about the pandemic with Dr Anthony Fauci, the nation's
leading infectious disease expert, but Newsom offered little additional information about the circumstances of the diagnosis.\n\nThe 1st case of the coronavirus variant in the US was detected in Colorado
on Tuesday [29 Dec 2020]. Experts have said it spreads faster than the common strain. Fauci said he wasn't surprised by reports of the latest case and suggested there are likely others in California and
other states due to international travel. 'I don't think that Californians should feel that this is something odd. This is something that's expected,' Fauci said. He added there is 'no indication that it
increases the virulence' of the infection any more than the variant that has been present in the US since February [2020]. Nor is there any evidence it can evade 'the protection that's afforded by the
antibodies that are induced by vaccines.'\n\n[byline: Vanessa Romo]\n\n--\ncommunicated by:\nClaudine Miller\nTulane Outbreak Daily\n<cmiller15@tuland.edu>\n\n----\n[A2] Florida, California\nDate: Fri 1
Jan 2021 4:50 AM CET\nSource: NBC [abridged, edited]\nhttps://www.nbcnews.com/news/us-news/covid-variant-found-florida-more-cases-identified-california-n1252637\n\n\nFlorida health officials said on
Thursday [31 Dec 2020] they have evidence that the COVID-19 variant first detected in the UK has been found in a patient there, making it the 3rd state to report the strain.\n\nAs the variant crops up in
new states, officials in San Diego County, California, said the strain, which is sometimes known as B.1.1.7., has been identified in 3 more people in San Diego County, bringing the number of confirmed
cases there to 4.\n\n[byline: Phil Helsel]\n\n--\ncommunicated by:\nProMED\n<promed@promedmail.org>\n\n----\n[B] China ex UK\nDate: Thu 31 Dec 2020 2:38 PM GMT+1\nSource: Yahoo News, Reuters report
[abridged, edited]\nhttps://news.yahoo.com/china-reports-first-case-coronavirus-133852730.html\n\n\nChina has detected its 1st imported case of the new coronavirus variant that is spreading rapidly around
Britain, according to a publication run by the Chinese Center for Disease Control and Prevention (CDC). The variant was detected in a 23 year old female student returning to China from Britain, who was
tested in Shanghai on [14 Dec 2020], according to the latest edition of China CDC Weekly published on Wednesday [30 Dec 2020].\n\nResearchers ran gene-sequencing on the patient's sample on [24 Dec 2020]
and found the strain is a variant known as 'VUI202012/01', the publication said. China will suspend direct flights to and from Britain indefinitely, a foreign ministry spokesman said last week. The patient
in Shanghai has been transferred to a designated medical institution for quarantine, and an investigation into her close contacts has started, according to the publication.\n\n[byline: Tom Daly; editing:
Hugh Lawson]\n\n--\ncommunicated by:\nProMED\n<promed@promedmail.org>\n\n----\n[C] Taiwan ex UK\nDate: Thu 31 Dec 2020 2:57 AM\nSource: The Seattle Times/The Associated Press (AP) report [abridged,
edited]\nhttps://www.seattletimes.com/business/asia-today-thailand-reports-250-new-coronavirus-cases/\n\n\nTaiwan has announced its 1st confirmed case of the new coronavirus variant that was found in the
UK. The patient is a teenage boy who returned from the UK on Sunday [27 Dec 2020] and is now receiving treatment, the Central Epidemic Command Center said on Wednesday [30 Dec 2020]. In response, Taiwan is
further tightening travel restrictions, requiring business travelers who previously could apply for shorter quarantines to do a full 14-day period and suspending international transfers at the airport from
```

To account for API monitoring and performance improvement, secondary log files were generated and stored in the backend. Whenever the API is accessed, a log file is generated and placed within the appropriate group as mentioned above (dependent on which endpoint is accessed) and stores the following information, as evidenced by the screenshot below:

- status of response
- request headers
- time accessed

- data source
- information returned from endpoint
- memory used
- time taken to fulfill request

Example: article id with correct parameters- 200 response



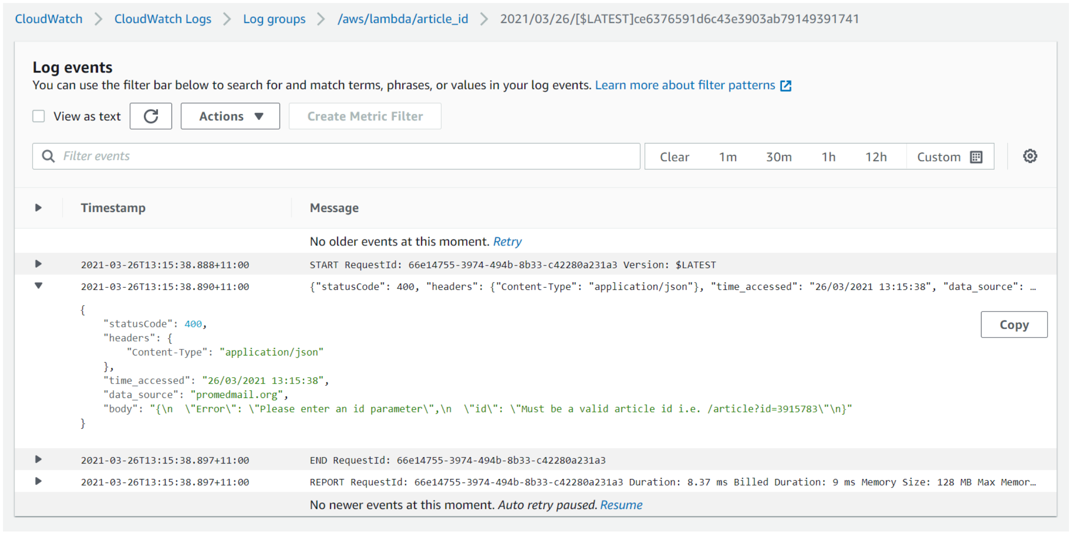
CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/article_id > 2021/03/26/[$LATEST]b2a096728a3142da97eb8282c94e6ed8

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                        Message
                                 No older events at this moment. Retry
2021-03-26T12:26:03.915+11:00    START RequestId: cc27e2f6-1c05-4fe7-9452-052799174575 Version: $LATEST
2021-03-26T12:26:05.135+11:00    {"statusCode": 200, "headers": {"Content-Type": "application/json"}, "time_accessed": "26/03/2021 12:26:03", "data_source…
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "time_accessed": "26/03/2021 12:26:03",
    "data_source": "promedmail.org",
    "body": "{\"articles\": [{\"archive_id\": 3915783, \"url\": \"https://promedmail.org/promed-post/?id=3915783\", \"date\": \"2016-01-06
17:49:25\", \"headline\": \"PRO/AH> Anthrax, human - Turkey: (E. Anatolia) cutaneous cases 2008-2014\", \"summary\": \"The 2014 monthly animal
outbreak pattern was January (5 livestock outbreaks), February (3), March (6), April (8), May (1), June (4), July (21), August (19), September
(18), October (6), November (3), December (1) -- not atypical in the northern latitudes -- for a total of 95 livestock outbreaks that involved 259
bovine cases, 49 sheep and goats, and 4 horses. A ProMED-mail post\\nhttp://www.promedmail.org\\nProMED-mail is a program of the\\nInternational
Society for Infectious Diseases\\nhttp://www.isid.org\\n\\n Agri had 2 livestock outbreaks in 2014; Bingol (0), Elazig (3), Hakkari (0), Igdir
(0), Kars (19), Tunceli (0), Van (1), and of the provinces that are mostly in the Eastern Anatolia Region: Ardahan (3), Erzurum (12), and Sirnak
(1). promed@promedmail.org>\\n\\n[During the period 2008-2014, there were 1123 human cases of anthrax in Turkey as a whole with 5 deaths, and just
150 human cases with one death. 30 (36.6 percent) patients were female, and 52 (63.4 percent) patients were male; ages were 18-69 and mean age was
43.77 13.05. In any year, multiple provinces suffer livestock outbreaks; in 2014, 37 provinces had one or more outbreaks. \\n\\n82 cases of
anthrax hospitalized at Ataturk University Faculty of Medicine, Department of Infectious Diseases and Clinical Microbiology in 2008-2014 were
examined retrospectively. In 2013, there were 34 provinces with cattle outbreaks and in 2012 31 provinces with bovine outbreaks. The most common
occupational groups were housewives (36.6 percent) and people working in animal husbandry (31.7 percent). CUTANEOUS CASES 2008-
2014\\n****************************************************************** So one can see that the risk of human anthrax in this region of Turkey is
very high, and it would not be inappropriate to describe the human appreciation of this risk as careless, and at a high cost. Communicated
by:\\nProMED-mail\\n<\", \"reports\": [{\"event_date\": \"1123-03-24 00:00:00 to 2021-12-24 00:00:00\", \"locations\": [{\"country\": \"Turkey\",
\"location\": \"Turkey\", \"geonames_id\": null}], \"diseases\": [\"anthrax cutaneous\"], \"syndromes\": [\"meningitis\"]}]}]}"
}
2021-03-26T12:26:05.137+11:00    END RequestId: cc27e2f6-1c05-4fe7-9452-052799174575
2021-03-26T12:26:05.137+11:00    REPORT RequestId: cc27e2f6-1c05-4fe7-9452-052799174575 Duration: 1221.17 ms Billed Duration: 1222 ms Memory Size: 128 MB …
```

Example: article id with missing parameters- 400 response



CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/article_id > 2021/03/26/[$LATEST]ce6376591d6c43e3903ab79149391741

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                        Message
                                 No older events at this moment. Retry
2021-03-26T13:15:38.888+11:00    START RequestId: 66e14755-3974-494b-8b33-c42280a231a3 Version: $LATEST
2021-03-26T13:15:38.890+11:00    {"statusCode": 400, "headers": {"Content-Type": "application/json"}, "time_accessed": "26/03/2021 13:15:38", "data_source": …
{
    "statusCode": 400,
    "headers": {
        "Content-Type": "application/json"
    },
    "time_accessed": "26/03/2021 13:15:38",
    "data_source": "promedmail.org",
    "body": "{\n  \"Error\": \"Please enter an id parameter\",\n  \"id\": \"Must be a valid article id i.e. /article?id=3915783\"\n}"
}
2021-03-26T13:15:38.897+11:00    END RequestId: 66e14755-3974-494b-8b33-c42280a231a3
2021-03-26T13:15:38.897+11:00    REPORT RequestId: 66e14755-3974-494b-8b33-c42280a231a3 Duration: 8.37 ms Billed Duration: 9 ms Memory Size: 128 MB Max Memor…
                                 No newer events at this moment. Auto retry paused. Resume
```

It is also to be noted that log files of a similar nature were created for the webscraper functions to ensure the maintenance and reliability of our product in the future, and further if any errors were to arise then it would allow for seamless debugging and performance improvement. For an example of how the WebScraper logs look please see GitHub repo:
SENG3011-WeByte/PHASE_1/API_SourceCode/Logs/

1.4 Project Plan

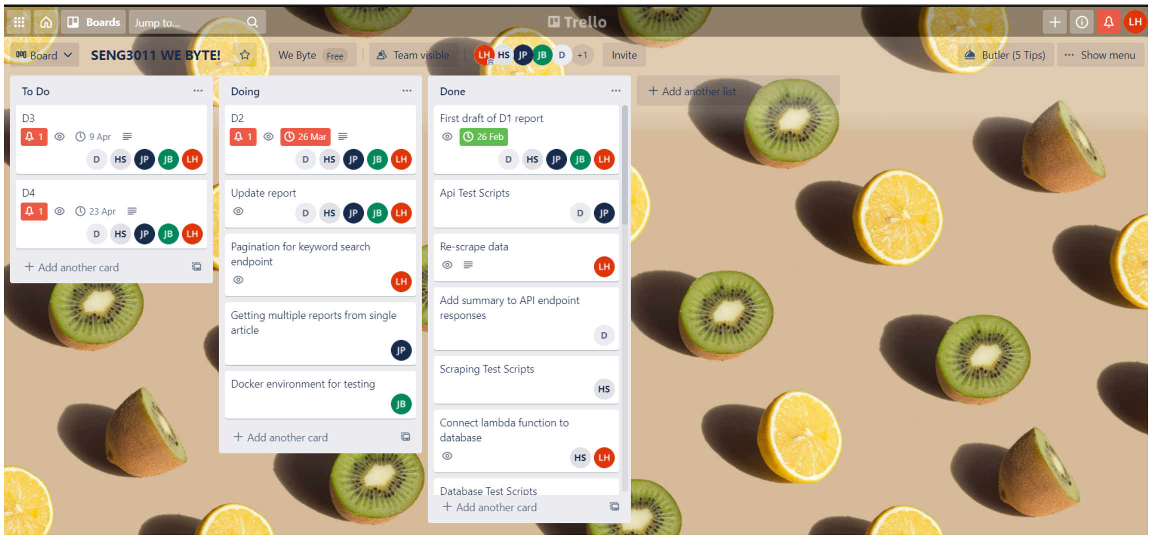
1.4.1 Communication and Meetings

In order to make sure everyone was on the same page, we conducted weekly meetings on Tuesdays at 1pm on Microsoft Teams, followed by a mentoring session at 1:40pm allowed us to clarify project deliverables/gain insight from our mentor about design choices. At the end of each Teams meeting we would take all the actions we had discussed and put them into our trello board (see 1.4.2).

Outside of these formal sessions, we used a whatsapp group for general communication and to allow team members to help each other out whenever they had questions. On top of this, we utilised pair programming sessions through screen share on teams to work through any particularly difficult tasks, or tasks that required the expertise of more than one team member at a time.

1.4.2 Trello Board

In order to best articulate team member responsibilities and plan out any deliverables and/or sprints, Trello, Teams(and in built Microsoft OneNote) and TeamGantt were used as all platforms allow for easy real time collaboration. Trello in particular was used for the management of sprints and the allocation of tasks amongst team members, whilst TeamGantt allowed us to ensure we meet all deadlines and complete the project requirements .

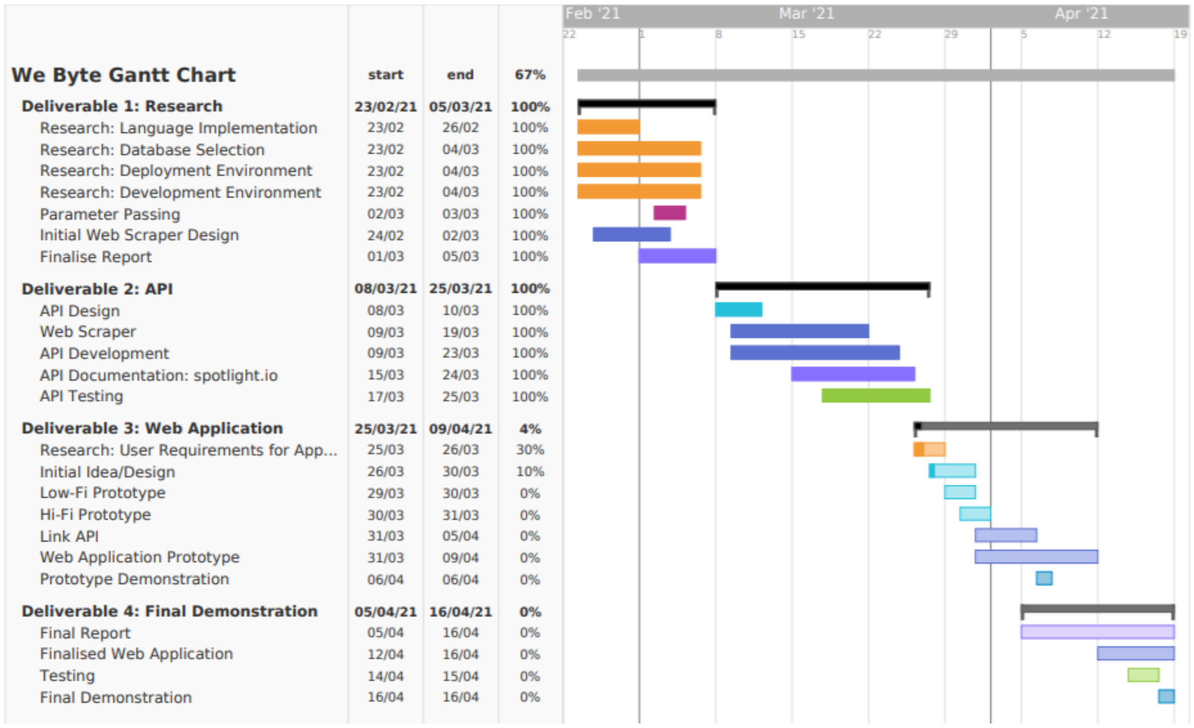


To Do
- D3 — 1 — 9 Apr — D HS JP JB LH
- D4 — 1 — 23 Apr — D HS JP JB LH

Doing
- D2 — 1 — 26 Mar — D HS JP JB LH
- Update report — D HS JP JB LH
- Pagination for keyword search endpoint — LH
- Getting multiple reports from single article — JP
- Docker environment for testing — JB

Done
- First draft of D1 report — 26 Feb — D HS JP JB LH
- Api Test Scripts — D JP
- Re-scrape data — LH
- Add summary to API endpoint responses — D
- Scraping Test Scripts — HS
- Connect lambda function to database — HS LH
- Database Test Scripts

1.4.3 Gantt Chart (link)



We Byte Gantt Chart	start	end	67%
Deliverable 1: Research	**23/02/21**	**05/03/21**	**100%**
Research: Language Implementation	23/02	26/02	100%
Research: Database Selection	23/02	04/03	100%
Research: Deployment Environment	23/02	04/03	100%
Research: Development Environment	23/02	04/03	100%
Parameter Passing	02/03	03/03	100%
Initial Web Scraper Design	24/02	02/03	100%
Finalise Report	01/03	05/03	100%
Deliverable 2: API	**08/03/21**	**25/03/21**	**100%**
API Design	08/03	10/03	100%
Web Scraper	09/03	19/03	100%
API Development	09/03	23/03	100%
API Documentation: spotlight.io	15/03	24/03	100%
API Testing	17/03	25/03	100%
Deliverable 3: Web Application	**25/03/21**	**09/04/21**	**4%**
Research: User Requirements for App...	25/03	26/03	30%
Initial Idea/Design	26/03	30/03	10%
Low-Fi Prototype	29/03	30/03	0%
Hi-Fi Prototype	30/03	31/03	0%
Link API	31/03	05/04	0%
Web Application Prototype	31/03	09/04	0%
Prototype Demonstration	06/04	06/04	0%
Deliverable 4: Final Demonstration	**05/04/21**	**16/04/21**	**0%**
Final Report	05/04	16/04	0%
Finalised Web Application	12/04	16/04	0%
Testing	14/04	15/04	0%
Final Demonstration	16/04	16/04	0%