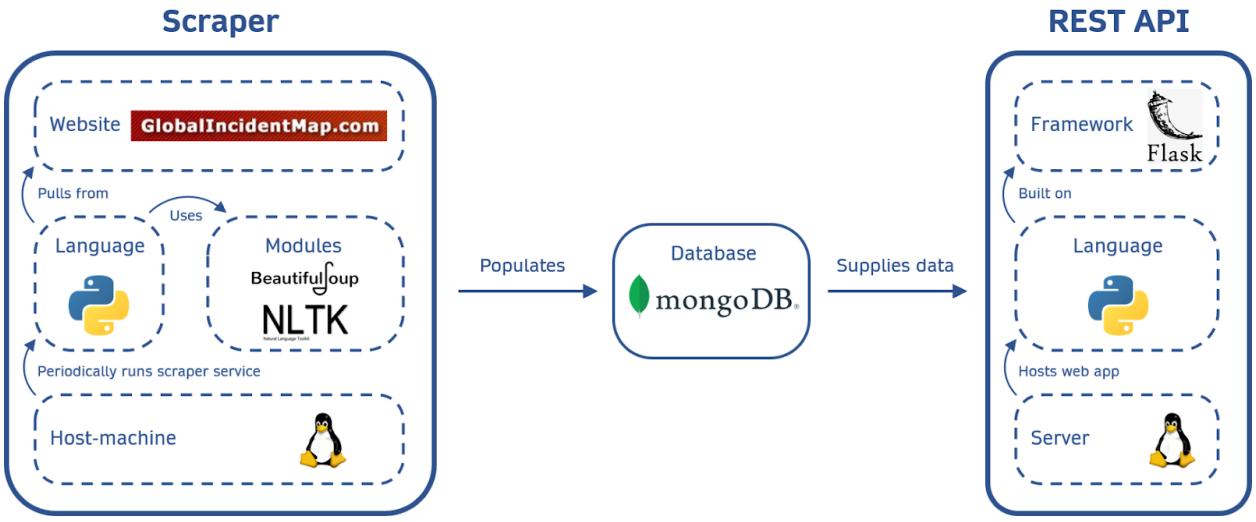
Design Detail

1 Overall Design

1.1 Web Scraper

For the web scraper we will be using Python due to its ease of use. Being an interpreted language, developing a webscaper should be quick and simple. The primary module we will use is BeautifulSoup which allows us to parse and search html pages with ease. We will use other libraries and modules like regex, word2number and Natural Language Toolkit to extract more detailed information.

The site we are scraping (globalincidentmap) offers detailed reports sorted in chronological order. Consequently the scraper will be run daily parsing only data it has not seen before. It may be the case that there is more information to be found in the articles it links to. If this is the case then we will have the scraper explore those articles as well searching for keywords.



1.2 REST API

The API module will be developed using Python with Flask, a web app development framework. Using this framework a simple web application can easily be created that allows users to submit data to an endpoint of the website using a HTTP request. Each request will include a parameter that specifies what data the response should include. The web application will use the parameter to filter the records in the database and return the appropriate results.

The web scraper explained in 1.1, will populate a MongoDB database that the API will access for the data it will send as a response. MongoDB will be used since it stores data in a JSON format, the same format used throughout the application, making it easier to interact with. This web application will need to be hosted on a server, such as on a CSE machine, to provide the ability to run in web service mode.

1.3 Swagger

To aid with the use of our API by other teams we have decided to use Swagger. Swagger allows for clear documentation of our API, in an easy to read form. It also allows for someone to create dummy calls to our API through the Try It Out feature. Thus, another person is able to see how the calls work and what form they should be in. This aids with understanding as they are easily able to see what the calls should look like, thus easy integrate our API into their web app.

1.4 Web App

1.4.1 Frontend

The web app is where we can showcase our API data, so it is important to choose a frontend tech stack that enables us to quickly create a working app, while at the same time allowing for complex component design that can show off our features.

The framework we chose to use was React. This is for two reasons. Firstly, many of us have industry experience working with React, and so we can write the website at a faster pace. React also allows for higher complexity with its component system, enabling our team to go above and beyond, without being hindered by the framework.

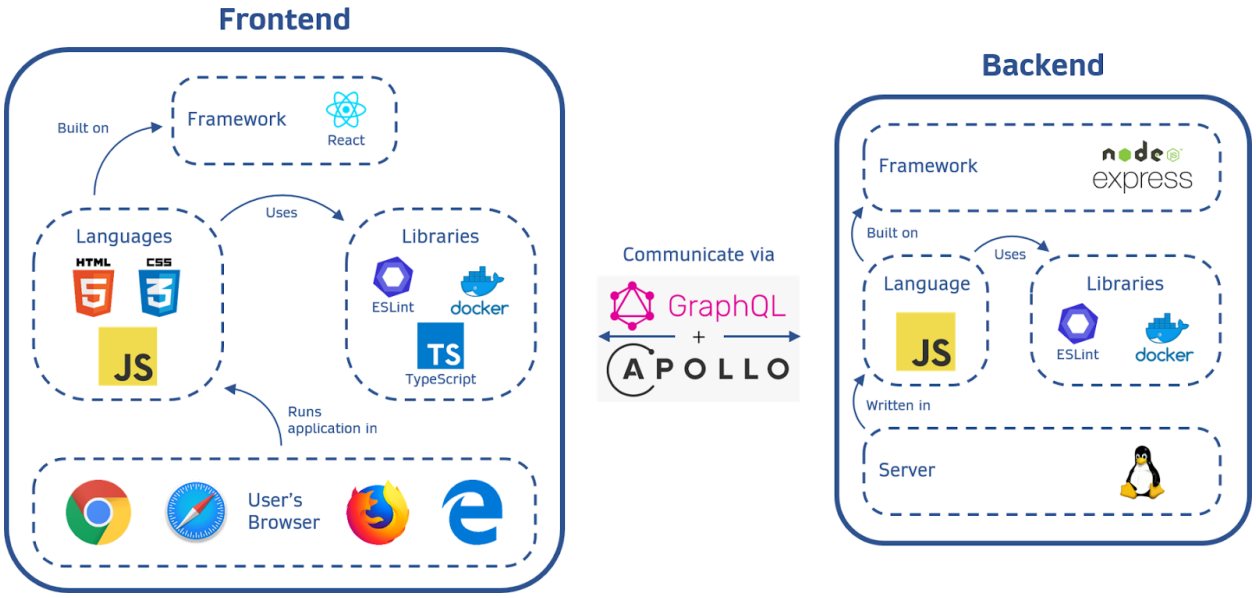
To communicate with the backend, we are using Apollo GraphQL. This is widely accepted to be the most straightforward way of communicating with the backend through a single endpoint, and allows our users to offload the processing of data to the database. This means that our website uses less power and loads at a faster rate than comparable websites using standard fetch requests. It also makes our endpoint much easier to use, as they only have to understand the schema for a single endpoint.

There are three supporting libraries that we are utilising: Typescript, ESLint and Docker.
Typescript allows us to typecast our javascript code, essentially reducing the time spent tracking bugs caused by javascript's dynamic typing system.
ESLint allows us to standardise our code style, reducing the time it takes for new developers to understand what is going on. It also does this automatically, so no extra time is spent on formatting.

Docker allows us to standardise the testing and development of the frontend. This means that everyone who runs our code should have identical outputs, allowing for easier bug fixing. It also automates a lot of the setup, allowing more devs to quickly become active in the development workflow when needed.

1.4.2 Backend

Our backend will be quite lightweight, as most complex analysis and data will be retrieved from our public API. Due to this, we decided to use NodeJS Express, which is known for its flexibility and ease of use. Express also has a GraphQL library which we can use to serve our services from a single endpoint. Just like the frontend, we will use ESLint and Docker. Docker in particular will allow us to quickly run a test backend and frontend instance in conjunction, improving our testing capabilities.



2 API Design

2.1 Endpoint Design

Clients can interact with the API through a single endpoint via HTTP requests. These requests will use parameters to contain all the necessary information required by the API to calculate and send back the appropriate results. Depending on the data contained in the parameter, the API will filter the data stored in the database and return the requested data as a JSON object. The parameter must adhere to a strict structure made up of 4 pieces of information represented in the form: {“start_date”, “end_date”, “key_terms”, “location”}, where start_date and end_date are in the form “yyyy-MM-ddTHH:mm:ss”. If the parameter does not adhere to the required structure

then the API will return an error result with an appropriate error code. Below is a table describing the possible response codes and the reason for them.

Response Code	**Reason**
200	Everything is ok and correct result is returned
400	There is an issue with the parameter
404	No data matching the parameter requirements could be found
405	The request method used is not allowed
500	The API had an issue processing the request

2.2 Example Interactions

Request Method	**Parameter**	**Response**

GET	{
	“start_date”:
	“2015-02-015T08:45:10”,
	“end_date”:
	“2015-11-01T19:37:12”,
	“key_terms”:
	“Brucellosis,Anthrax”,
	“location”: “sydney”
	}
		{"date": "2015-02-15 17:33:00", "country": "Australia", "url": "http://www.abc.net.au/news/2015-02-16/pig-brucellosis/6116970", "coord": "-33.864, 151.205", "city": "New South Wales", "disease": "Brucellosis", "description": "[ABC.net.au]\u00a0 AUSTRALIA :: Call for pig dogs to be tested for brucellosis\n\r\n\"Veterinarians in the Hunter Valley and North West regions of New South Wales are calling on pig dog owners to be be on the lookout for signs of brucellosis after several dogs tested positive for the disease.\"\n\r\nRead Full Article At :: http://www.abc.net.au/news/2015-02-16/pig-brucellosis/6116970"}

GET	{ “end_date”: “2015-11-01T19:37:12”, “key_terms”: “Brucellosis,Anthrax”,	400 status code with no JSON object returned due to the parameter not meeting the required structure

	“location”: “sydney” }	

GET	{
	“start_date”:
	“2015-10-01T08:45:10”,
	“end_date”:
	“2016-11-01T19:37:12”,
	“key_terms”: “Coronavirus”,
	“location”: “sydney”
	}
		404 status code with no JSON object returned due to the API not having any data that meets the parameter requirements

POST	{
	“start_date”:
	“2015-02-015T08:45:10”,
	“end_date”:
	“2015-11-01T19:37:12”,
	“key_terms”:
	“Brucellosis,Anthrax”,
	“location”: “sydney”
	}
		405 status code with no JSON object returned due to the request method not being GET

GET	This could happen independent of the supplied parameters, as a 500 status code is a catch for uncaught errors in the server.	500 status code with no JSON object returned due to an API error, possibly due to a missed edge-case

3 Implementation Choices

3.1 Implementation Language

The main languages that we have decided to go with are Python and JavaScript.

3.1.1 Python

We are using Python for the web scraper and RESTful API due to the simplicity of the language. It allows us to focus on the implementation of the services, rather than getting bogged down in the complex structures that other languages, such as C or an Object Oriented language such as Java, poses. This also helps to increase the reliability of our code, which allows for seamless collaboration throughout the team.

Also, Python allows us to use a framework such as Flask, which handles a large amount of the overhead for creating a Web API. This allows us to yet again focus on the implementation of the API, increasing our productivity.

We understand that the choice to go with Python will reduce the speed of our program, as Python is inherently slower than a language like C.However, the benefits of increased productivity and readability outway the negative performance effects.

3.1.2 JavaScript

We have decided to use JavaScript, coupled with HTML and CSS, as our main frontend language as it has become an industry standard for web applications. Also, we are able to use frameworks such as React to help with the development of our frontend. React offers standard conventions and structuring that will help in the continued development of our web application. It will allow us to add new features and pages with ease, in a way that appears integrated with the rest of the site.

We will also be using NodeJS as our backend for the web application, coupled with Expressed, as this is a well used tech stack. The simplicity of Express will allow for the backend to be created simply and efficiently, and will also help in adding extra features as the project goes on.

3.2 Development and Deployment Environment

For our deployment environment we have decided to go with Linux. The reason we are using Linux is that it is lightweight and simple to set up for the small services that we are creating. Also, it is a common choice for services like ours so there is a large amount of documentation and help online, so we will be able to easily setup the services and diagnose issues when they arise.

However, all our frameworks work cross platform so during development we can all use them independent of our own environments.This speeds up development, as we do not have to individually configure our environments to help run this process. And with the introduction of Docker we will be able to perform the setup for our environments quickly.

3.3 Libraries

3.3.1 Typescript

Typescript is an open source library that adds typing to traditional javascript. This improves clarity, as a developer can discern more information from any given variable, and also reduces room for error when variables are misinterpreted. The overarching reason we added typescript was to improve our quality, and pace, of code written.

3.3.2 Apollo

Apollo is the industry-standard GraphQL implementation for services written in javascript. Instead of using a traditional REST service, we can access the backend through a single endpoint, allowing our users to offload most computation to our services. This means that our app uses less power and loads at a faster rate than comparable websites using standard fetch requests. It also makes our endpoint much easier to use, as they only have to understand the schema for a single endpoint.

3.3.3 Beautiful Soup

Beautiful Soup is an open source library which allows aids in the extraction of data from HTML files. It creates a parse tree of the file which can be easily navigated, searched and modified in idiomatic ways. This is incredibly useful for scraping data off websites as it grants the ability to use extract data based on its HTML structure.

3.3.4 Natural Language Toolkit (NLTK)

NLTK is a collection or libraries and programs which help programs reason about natural language texts. These include tagging, machine learning, parsing, probability, tokenization, stemming and chunking of texts. NLTK will be used to find more detailed information from the articles and descriptions.

3.3.5 Requests

An open source library that makes sending HTTP requests very easy, abstracting most of the required functionality required with useful methods. Using this library's 'get' method along with the API's URL and a JSON object parameter, a HTTP GET request will be sent to the API and a data object returned that can be converted into a JSON object using the library's 'json' method, all in one convenient line.

3.3.6 Flask-PyMongo

Flask-PyMongo is an open source library that bridges Flask and PyMongo and provides some convenience helpers. PyMongo provides the connection to the MongoDB database used by the API.