Epidemic Eagle

Prepared for SENG3011

Created by Aaron Shek, Sahibpreet Bassi, Daniel Steyn, Liam Treavors and Mihail Georgiev

Context

In recent times, the COVID virus has affected the entire world, due to the rapid infection rate of the virus. But since vaccines have been developed, minimal effort has been made to prevent future outbreaks.
One solution to this issue is to notify government facilities when a potential outbreak is predicted to happen. Resources could then be arranged and handed to medical centres to prepare and reduce the spread of an outbreak.
One method to achieve this solution is to read news articles about reports on diseases, and broadcast a message if patterns of an infection around an isolated area exist.
Although manually web searching reports would solve this issue, it would be very labour intensive and time-consuming to collect and process all the data. That is why we have designed an API that will automate and summarise these articles and reports into uniform data which can be viewed easily.

1. API Model

1.1. Routing

For the routing of the API, we have thought about which routes to provide to the user. GET requests send information such as article bodies and will be a required route. But other requests such as POST and PUT are not included, since users will not need these functionalities. Additionally, admin routes will not be needed since we plan to generate and provide data for our API automatically through the use of a web scraper.

1.2. Web Scraper

For our API the web scraper will process numerous websites such as ProMed, then collate the relevant data and return it to the user. The web scraper will also automatically update or delete outdated reports since it continually rechecks websites. Since a web scraper is used for our API, this means POST, PUT and DELETE requests will not be included.

1.3. Caching Data

Web scraping is a powerful tool, but has the drawback of high computational time. So if the web scraper is called every time a GET request is made to the API, it would also be a pain point to the user experience. This is why our team has decided to cache data from the web scraper to send to the user, and instead intervally run the web scraper. This will, in turn, lead to rapid interactions with our API which benefits the user. However an issue with cached information is outdated information. The cache will be updated every 4 hours from the web scraper, to ensure the information is up-to-date.

1.4. Response and Storage Objects

Data from the web scraper will be stored in a database, as a JSON Object. This was favourable over such types as XML and due to compatibility with programming languages.
Data that is sent from our API will also be in a JSON format, with a supporting status code such as 200 or 404. This was done since it reflects the industry standards and conventions.

1.5. Pagination / Filtering

An issue that affects users is the heavy loading of a singular route, such as getting all reports from a certain date. That is why our API will include pagination to reduce the load on the user as well as server usage. Our current plan is to limit results to 10 items, which then page_number will be passed as a path parameter to access further pages.
Filters such as keywords and symptoms of a disease will also be incorporated, for similar reasons stated above.

1.6. Versioning the API

Our API will continually be developed and pushed in 3 phases.
However if our API is updated, some functionalities of old routes may be preferred over new versions of the same route. This is why our API will include different versions of our API, via changing the route call such as “/v1/api/” to “/v2/api/”.

1.7. Web Hosting

Although our team is very experienced with creating an API, web hosting was uncounted by our team. We decided to choose an external service which hosts servers instead of personal local hosting to reduce costs.

2. Design Details

2.1. Parameters

For our API we will use path parameters to pass in information such as filters.This is comparable to a JSON body, but errors will be easier to detect in a path compared to body. The internals of our API are described below.

Route	Method	Params (required)	Optional Params	Response	Definition

/api/articles	GET	start_date: <string::date>
		end_date: <string::date>
		key_terms: <string>
		location: <string>
			page_number: <string>	{
				articles : [list of 10 <object::article>s],
				num_pages : 1,
				page_number : 1,
				}
					gets all articles in the dates submitted
					can be filtered by key_words and location

/api/articles/{:article_id}	GET	article_id:<integer>		{<object::article>}	gets article data (all reports from an article)

/api/reports	GET	start_date: <string::date>
		end_date: <string::date>
		key_terms: <string>
		location: <string>
			page_number: <string>	{
				reports : [<object::report>],
				num_pages : 1,
				page_number : 1
				}
					gets all reports from dates.
					can be filtered by key_words and location

/api/reports/{:report_id}	GET	report_id:<integer>		{<report>}	gets singular report data

/api/search	GET	start_date: <string::date>
		end_date: <string::date>
		key_terms: <string>
		location: <string>
			page_number:<string>	{
				results: [list of 10 <object::search>s],
				num_pages : 1,
				page_number : 1,
				}
					gets shortened results in the dates submitted.
					can be filtered by key_words and location

Model	JSON Object

article	{
	url: <string>,
	date_of_publication: <string>,
	headline: <string>,
	main_text: <string>,
	reports: [<object::report>]
	}

report	{
	diseases: [<string>],
	syndromes: [<string>],
	event_date: <string>,
	locations:
	[
	country:<string>,
	location:<string>
	],
	}

search	{
	article_id:<string>,
	url: <string>,
	date_of_publication: <string>,
	headline: <string>
	}

2.2. Result Gathering

We will run the web scraper at a constant time interval every 4 hours. At each interval, the web scraper will read all articles from multiple websites such as ProMed, and break it down into articles and reports based on keywords within the body of text. It will then store these articles and reports within our Postgres database. Articles will store the main body of text, the headline of the original article, the url and the publication date of the article, while reports will store the disease, the location, the syndromes and the reported date of the illness. Reports will also link to the article they originated from, and can only link to that one article.

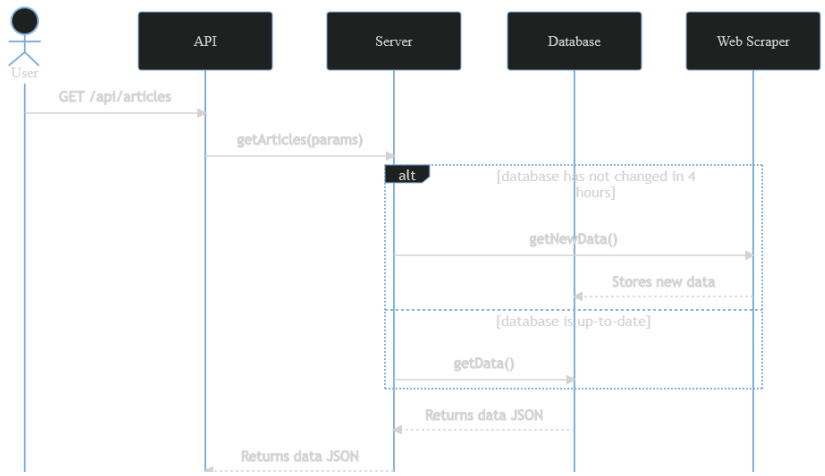
The user of our API won't see any of this scraping however, when they are searching for certain diseases, the API will query our database, and find matches on disease, location and time periods, without interacting with ProMed itself. This will greatly reduce waiting time for users, as searching a database will be much quicker than scraping through articles and then comparing key terms. Matching entries in the database will then be returned to the user as a list of JSON Objects, with each Object being the requested type of data, such as an article or report.

2.3. Sample HTTP Calls

In order to visualise our API, sample HTTP calls have been added below.

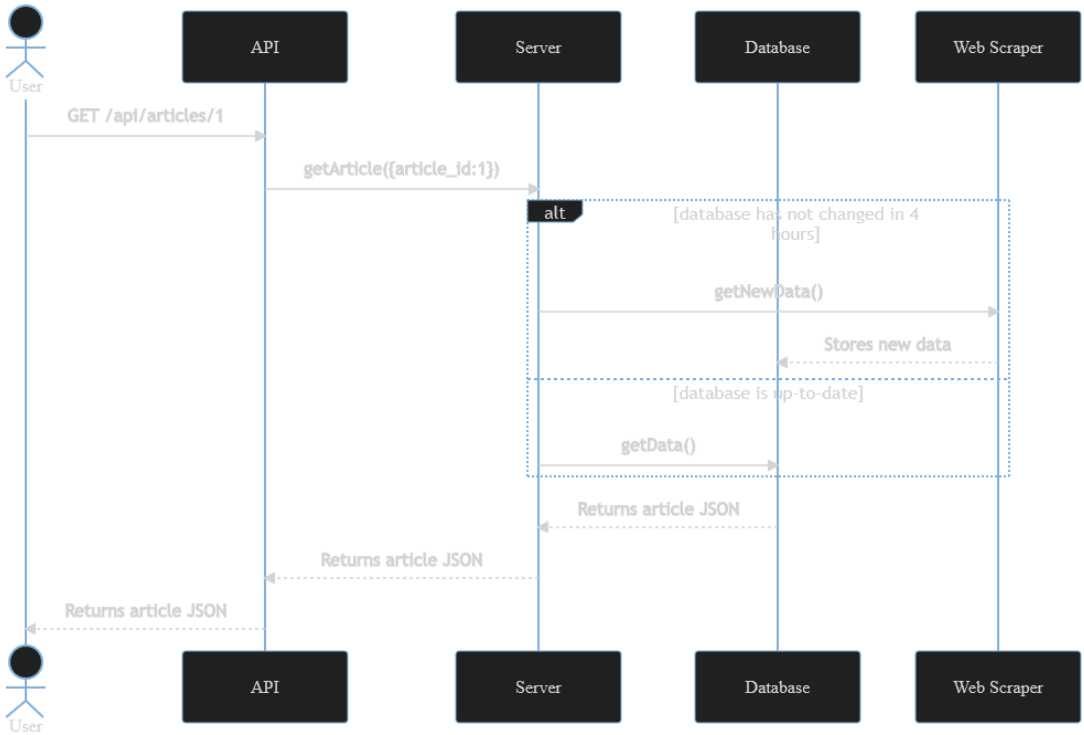
Request :
/api/articles/?start_date=2018-12-12&end_date=2018-12-13&key_words=influenza&location=Vietnam
Response:


```
{
  "articles": [
              {
               "url": "www.who.int/lalala_fake_article",
               "date_of_publication": "2018-12-12",
               "headline": "Outbreaks in Southern Vietnam",
               "main_text": "Three people infected by what is thought to be H5N1 or H7N9  in Ho Chi Minh city. First infection occurred on 1 Dec 2018, and the latest is reported on 10 December. Two in hospital, one has recovered. Furthermore, two people with fever and rash infected by an unknown disease.",
               "reports": [
               {
                "event_date": "2018-12-01 to 2018-12-10",
                "locations": [
               {
               "country": "Vietnam",
               "location": "Ho Chi Minh"
               }
                ],
               "diseases": [
               "influenza"
                ],
                 "syndromes": [
               "Acute fever and rash"
                ]
               }
              ]
              }
  ],
  "page_number": 1,
  "page_count": 1
}
```

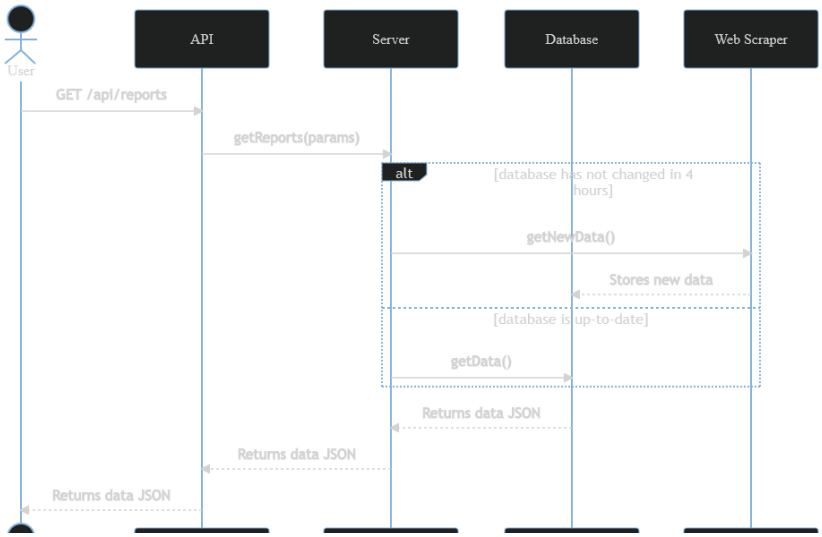
Request : /api/articles/article_id=1
Response:


```
{
 "url": "www.who.int/lalala_fake_article",
 "date_of_publication": "2018-12-12",
 "headline": "Outbreaks in Southern Vietnam",
 "main_text": "Three people infected by what is thought to be H5N1 or H7N9  in Ho Chi Minh city. First infection occurred on 1 Dec 2018, and the latest is reported on 10 December. Two in hospital, one has recovered. Furthermore, two people with fever and rash infected by an unknown disease.",
 "reports": [
   {
    "event_date": "2018-12-01 to 2018-12-10",
    "locations": [
      {
       "country": "Vietnam",
       "location": "Ho Chi Minh"
      }
    ],
    "diseases": [
     "influenza"
    ],
    "syndromes": [
     "Acute fever and rash"
    ]
   }
 ]
}
```

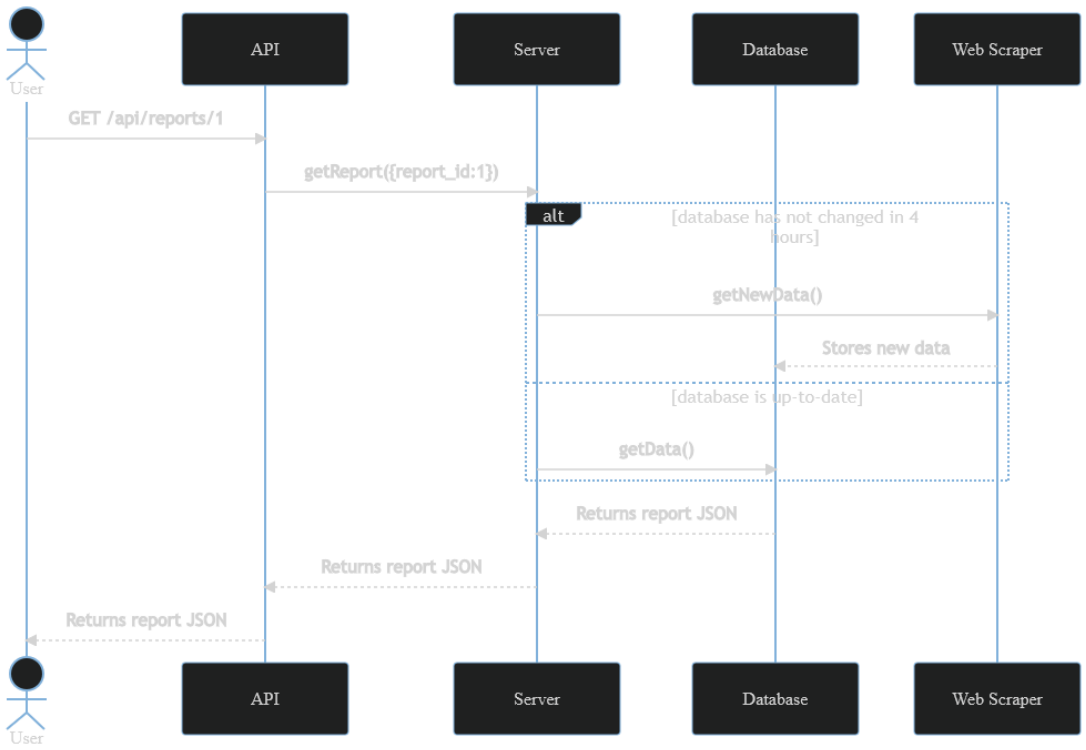
Request :
/api/reports/?start_date=2018-11-12&end_date=2018-12-12&key_words=ebola&location=Democratic Republic of Congo Kivu
Response:


```
{
 "reports": [
            {
            "event_date": "2018-11-13 to 2018-11-20",
            "locations": [
             {
              "country": "Democratic Republic of Congo",
              "location": "Kivu"
             }
            ],
            "diseases": [
             "Ebola"
            ],
            "syndromes": [
             "Fever and headache",
             "Abdominal pain",
             "Unexplained haemorrhaging"
            ]
            },
            {
             "event_date": "2018-12-04 to 2018-12-10",
             "locations": [
             {
              "country": "Democratic Republic of Congo",
              "location": "Kivu"
             }
             ],
            "diseases": [
             "Ebola"
            ],
            "syndromes": []
            }
 ],
 "page_number": 1,
 "page_count": 1
}
```

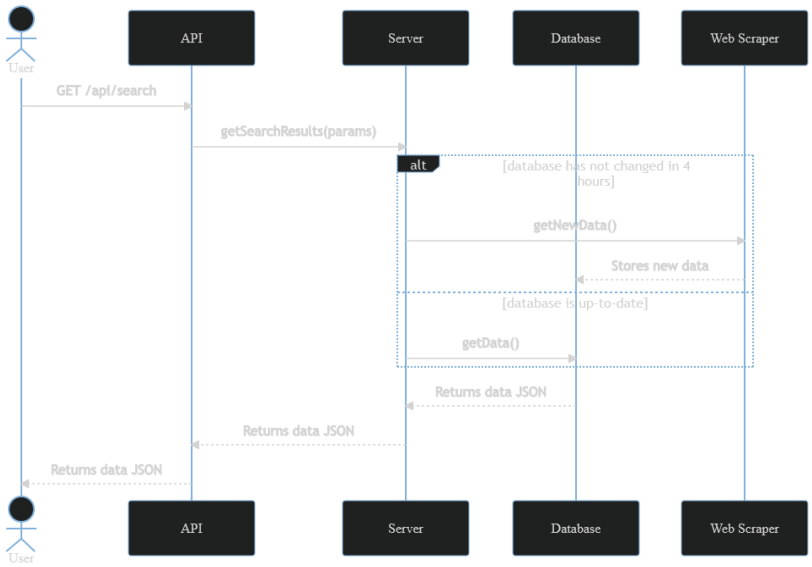
Request : /api/reports/?report_id=1642
Response:


```
{
 "event_date": "2018-11-13 to 2018-11-20",
 "locations": [
        {
        "country": "Democratic Republic of Congo",
        "location": "Kivu"
        }
 ],
 "diseases": [
        "Ebola"
 ],
 "syndromes": [
        "Fever and headache",
        "Abdominal pain",
        "Unexplained haemorrhaging"
 ]
}
```

Request :
/api/search/?start_date=2019-02-10&end_date=2022-02-10&key_words=measles&location=Afghanistan
Response:


```
{
 "results": [
        {
        "article_id": "1809",
        "url": "www.who.int/emergencies/disease-outbreak-news/item/measles-afghanistan-paktya",
        "date_of_publication": "2022-01-08",
        "headline": "Measles Outbreak in Paktya"
        },
        {
        "article_id": "1756",
        "url": "www.who.int/emergencies/disease-outbreak-news/item/measles-afghanistan-balkh",
        "date_of_publication": "2021-08-14",
        "headline": "Measles Outbreak in Balkh"
        },
        {
        "article_id": "1429",
        "url": "www.who.int/emergencies/disease-outbreak-news/item/measles-afghanistan-kunduz",
        "date_of_publication": "2019-05-27",
        "headline": "Measles Outbreak in Kunduz"
        },
        {
        "article_id": "1398",
        "url": "www.who.int/emergencies/disease-outbreak-news/item/measles-afghanistan-zabul",
        "date_of_publication": "2019-04-18",
        "headline": "Measles Outbreak in Zabul"
        }
 ],
 "page_number": 1,
 "page_count": 1
}
```

3. Development, Implementation and Deployment

3.1. API's Development

For our EpidemicEagle API's development, our team has chosen the following for its tech stack:

- The 'Scrapy' web crawling framework
- The 'PostgreSQL' RDBMS
- The 'FastAPI' web framework
- Heroku, a cloud service platform
- The Linux Operating System

3.1.1. Scrapy

Scrapy is a web crawling framework that is written in Python. It was built to extract data from a webpage as a 'general purpose web crawler', and additionally can be used to extract data using APIs. After extracting the data we want, we can then process and store the data in our own preferred structure and format.
Scrapy carries out these operations through the use of 'Spiders'. These are essentially classes which define how a certain site (or a group of sites) will be scraped. The key feature is that they are self contained objects when in use. The user can define custom behaviour or crawling or parsing pages for individual websites.

Despite there being numerous options for web scraping, including Beautiful Soup, Selenium, ParseHub, ScrapingBee, SCRAPEOWL, Jsoup, Gumbo and so on, Scrapy was the better choice for our team for the following reasons:

- For our team it is more accessible because it is written in Python, which is known to be an easily learnable, very readable and powerful language.
- In comparison to Selenium and Beautiful Soup, Scrapy is the best for dealing with large projects where efficiency and speed are required.
- It is also asynchronous, unlike the other two, meaning it can make requests in parallel rather than one at a time, and it is memory and CPU efficient as a result.
- It is a 'complete' web scraper framework with built in crawling, whereas Selenium for example is a parsing library online that manually needs to be built in an infinite loop for crawling.

Because Scrapy is a 'complete' framework, it offers more features that most other scrapers which as libraries, do not have;

- Scrapy uses extended CSS selectors and XPath expressions for extracting data from HTML or XML sources
- Scrapy has an interactive shell console to scrape data, which can be used to experiment with CSS and XPath expressions to scrape data
- Scrapy can generate feed exports in multiple formats (JSON, CSV, XML) and storing them in multiple backends (FTP, S3, local filesystem)

The only drawback this scraper may have is it's learning curve, but Scrapy is clearly powerful enough to outweigh this.

3.1.2. PostgreSQL

PostgreSQL is an advanced, enterprise class, relational database management system that has been backed by 20 years of development. It is free, open source, and focuses on being extensive and complying to SQL. It provides a relational engine which is capable of efficient implementation of relational operations, backup and recovery, transaction processing for concurrent access, a powerful query optimizer based on genetic algorithms, handles JSON as a native data type and allows user defined data types. Many web, mobile, geospatial, and analytics applications use it to this day as their primary source of data, especially as a robust back end that powers many dynamic websites. It supports many languages, including Python which is one of the key languages we will be relying on for this project.

The first clear advantage of using PostgreSQL is that our team has very in-depth knowledge and experience about this system, as we have used it in the past. This means there is less to learn and time can be better allocated to other tasks. Additionally, as all members have experience with the language, troubleshooting becomes a much more efficient endeavour. Comparing PostgreSQL to other database options reveals that it is feature rich, explored below;

- PostgreSQL is SQL compliant, and implements 160 of the 179 core features of the SQL standard, unlike MySQL for example, which is only partially compliant. Additionally, PostgreSQL is very well documented and any operations that deviate from the standard are explained in detail,

- PostgreSQL is the appropriate when large systems are in question, data needs to be authenticated, and read/write speeds are important,

- PostgreSQL is the better choice for projects revolving around complex procedures, integration, intricate designs, and data integrity.
    - It can handle complex queries and huge databases compared to MySQL, and is usually the preferred solution to complicated, high volume database design. PostgreSQL can additionally define data types, index types, functional languages, and NoSQL. The fact that it is a 'relational' database means the data inserted can relate to one another, unlike other NoSQL databases (e.g. MongoDB). "

3.1.3. FastAPI

FastAPI is a web framework built on Python, for developing RESTful Web APIs. It is built on the Asynchronous Server Gateway Interface (ASGI) specification, which is a spiritual successor to WSGI. It was invented to provide a standard interface between Python web servers, frameworks and applications that were async capable. This means that, if many requests are incoming, requests do not have to wait for others to complete before they are processed.

The FastAPI framework additionally generates the documentation when the application is run. This is helpful because in most API builds, Swagger UI or Redoc needs to be written separately for documentation about endpoints and types.

The key positives that helped our team's decision to choose FastAPI, compared to Django or Flask, are listed below. FastAPI:

- is based on the Pydantic library also built on Python, for parsing and validating data (which Flask does not have). This is useful when it comes to enforcing type hints at runtime.
- has high performance speeds that are comparable to NodeJS and Go, fast to code,
- has great editor support,
- is easy to use and learn,
- Robust
- is based on the open standards for APIs.
- Faster than Flask
- has flexibility code wise, and doesn't restrict the code layout, unlike Django.

3.2. Python

Upon observing our tech stack, it is clear that two components use Python. This was an intentional choice, because of Python's unique advantages compared to other languages.

Python is often the first choice language for many programmers and students because:

- it is in high demand in the industry environment,
- Python is a flexible language in the sense that it does not restrict developers from developing a certain sort of application,
- It is the second most popular used tool for data science and analytics
- There are hundreds of of Python libraries being used in a variety of projects
- It is an efficient, reliable and fast language, that can be used in multiple environments such as mobile and desktop applications, web development, hardware programming and more,
- Python has a huge community for support
- It is extremely easy to learn and use.

Both Scrapy and FastAPI are Python based frameworks. Part of our team's decision to choose these technologies is because our team is experienced with the language, and we can focus more on producing a product rather than spending hours learning the basics and setup.

Python also proves useful when working with JSON data. Compared to languages like Ruby on Rails or other web hosting languages, Python has inbuilt functions that can treat dictionaries like JSON Objects and vice versa. Consequently our team can reduce the time spent coding and more time on testing and designing an end product.

3.3. API Deployment

We decided to compare two main web hosting servers, being AWS and Heroku. The comparison can be seen in the following table.

Article	Heroku	Amazon Web Services (AWS)
Service	Platform as a Service (PaaS)	Infrastructure as a Service and Platform as a Service (IaaS)
Initial Production	Infrastructure is provided by Heroku and fast to deploy.	AWS requires manual setup of the environment, being a relatively complicated process.
Traffic	Heroku can handle small/medium amounts of traffic.	AWS scales with internet traffic and can handle very high loads.
Cost	The cost of hosting on Heroku is average, and increases greatly for large applications.	AWS is extremely cost effective, as the service is a PAYG model.
Monitoring	Includes an application monitoring system with error responses.	No monitoring system.
Database Management	Heroku has an extension named "Heroku Postgres" for database	Databases are manually set up by the user, but can be chosen to fit the application.
Requirements	Heroku provides a ready-to-use environment.	AWS requires prior knowledge of Unix.

Ultimately, our team decided to go with Heroku as our web hosting service. It is not a perfect solution as it is known for high inbound and outbound latency, lower network performance and it is less suited for heavy-computing. Given our circumstances, however, we feel it is the best option for

our project. Creation of a server is a much simpler process than on AWS, it offers rapid deployment with a ready-to-use environment, minimal downtime during system updates and a more startup-friendly learning curve. It also provides a monitoring system and a database manager. Heroku requires no infrastructure maintenance, which is very desirable for such a small team as we can dedicate more team members to development tasks. Given all the advantages listed, we are willing to take a slight hit on performance as we feel we will be able to produce a better product with Heroku as we can focus more on development.