Final Report

Epidemic Eagle

Final Report

Prepared for SENG3011

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

- Epidemic Eagle
  - Final Report
- Project Introduction and Overview
  - Introduction
  - Our Solution
    - Scope
    - Solution
    - Business Value
    - API Model
  - Overview of Features
- Use Cases and Requirements
  - Overview
  - Use Case 1: Travel Agency
  - Use Case 2: Traveller
  - Use Case 3: Normal User
  - Requirements of the API
- Design Details
  - API Model
  - Web Scraper
  - Caching Data

        - Response and Storage Objects
        - Pagination / Filtering
        - Versioning the API
        - Web Hosting
    - Parameters
        - Web Scraper
            - Web Scraper Example
        - Web Scraper Results
        - Sample HTTP Calls
            - Side Note
    - Development, Implementation and Deployment
    - API's Development
        - Database System
        - Web Hosting Services
        - Programming Language
    - Implementation and Software Architecture
        - Justification of Software Architecture
            - Back End Software
            - Scrapy
            - FastAPI
            - Heroku PostgreSQL
        - Front End Software
            - Bootstrap
        - External APIs
            - Disease.Sh API
            - TravelAdvisory API
            - teamViral API
    - Further Justifications
        - PyTest
    - Challenges addressed and shortcomings
        - Heroku and Heroku PostgreSQL coupling
        - Heroku Timeouts
        - Shortcoming of web scraping with Dynamic Websites
        - Shortcoming of PostgreSQL
    - Key Achievements and Benefits of Our Software
        - Heroku Hosting
        - Front End
        - Back End
    - Team Structure
    - Software Tools
        - Jira
        - Confluence
        - GitHub
        - Discord
        - Availability Tool: "When2meet"
        - Google Docs
        - Microsoft Teams
    - Deliverable Workload
        - Deliverable 1
        - Deliverable 2
        - Deliverable 3
        - Deliverable 4
    - Reflection
        - Imbalanced Tasks
        - Departed Member
        - Jira Task Delegation
        - Teamwork
        - Future Practices
    - Conclusion

Project Introduction and Overview

Introduction

COVID-19 has been an eye-opening experience for society, showing how disease outbreaks affect the modern world. However post-COVID, society is regaining confidence for the outside world due to a change in disease awareness including vaccinations and ready-made COVID tests at home. One particular industry that is and will continue to generate revenue is the flying industry. According to Sydney Transport, the flight industry is seeing a 'steady recovery in domestic passenger numbers', as 'more than 82,000 passengers' were expected on the Easter Weekend.

However passengers are still worried about their personal safety when flying. In a 2020 survey, made by IATA, at least 79% of passengers were very concerned or somewhat concerned with visiting foreign countries, including the airport, the hotel and tourist attractions.

Some of these concerns stem from the pain point of the lack of information. In an interview with the general public, we learned that it is difficult for them to sift through multiple search results surrounding the location they're visiting. This concern ranges from finding results of medical articles to news reports. Additionally, some people in the survey had trouble finding past articles without in depth searching or loading multiple results.

Our Solution

Scope

Through our research and user interviews, we knew what our solution needed to be at its core - a reliable search engine dedicated to disease information. But we wanted to expand this even further to provide a unique solution in the market. To do this, we first had to revise our target audience scope. In order to properly cater to travellers, we also had to cater to travel agencies as they handle a large proportion of the travel industry. This change of scope was reinforced by a survey done by the IATA (International Air Transport Association), which stated that 43% of those interviewed would prefer to use a "travel agency, travel management company or corporate travel department" to book their flights.

Solution

Thus our solution is a hub for centralised information, providing travellers all the information they need to be certain in their decision making and everything travel agencies need to keep their clients informed. The platform is built around our search engine - a dedicated source for relevant disease information with multiple levels of specificity for users to search by. The platform itself provides tools to assist agencies in handling their many clients - as well as attract new clients - and an automated system to keep them informed on new reports relevant to their clients.

Business Value

In terms of business value, we provide value to our independent customers by being a reliable source of disease information worldwide and we provide value to organisations (travel agencies) by connecting them to new clients and allowing them to better manage their current clients. As a business, we will generate revenue through a freemium model, whereby some features will be locked behind a paywall. Additionally, we will run advertisements on our free versions.

API Model

To acquire the information needed for travellers about diseases and past articles would require a large amount of manual labour manually searching through reports. This is why we decided to work on creating an API to eliminate manual labour and save time for travellers. The API has been designed to automatically collect articles and reports from disease information websites and summarise the information into a format that can be quickly and easily viewed by the user.

Overview of Features

Our software provides many features for a range of audiences. We have decided to categorise the features of our software into three use cases.

For a travel agency our software provides:

- Login System
- Customised Dashboard
- Communication Between New and Existing Clients

It also provides to single travellers for:

- Customised Dashboard
- Trip Information and Disease Reports
- Communication to Agency

For any user they can do the following:

- Inquire for a Quote
- List of Searches
- Search for Location
- Search for Reports
- Search for Articles

Our API Model has the features:

- Search for articles by dates, location and disease
- Search for reports by dates, location and disease
- Search for posts by dates, location and disease
- Search for article by id
- Search for report by id

The following features help our target audiences by synthesizing information into a single website, reducing the need to search up multiple websites to keep themselves safe from disease warnings. These features are targeted towards the following two types of users: individuals who are travelling to other countries and travel agencies looking to advise travellers. Our application provides features to help both users safely and easily find the information they need to make more informed travel decisions.

Our platform also allows both travellers and travel agencies to login to a more personalised dashboard. The dashboard page allows travellers to plan out their trips and save locations they intend to visit allowing for easy access to information relevant to them. They can also access any disease reports that are in locations they are interested in. For travel agencies they are presented with a different dashboard more tailored towards communication with their clients. From here they can easily inform their clients of any disease reports in locations relevant to the specific client.

Use Cases and Requirements

Overview

Use cases and requirements are important to demonstrate why our software platform has included features, to apply to the target audience and their pain points or specific needs.

This section shows the front end implementation of the features we have detailed above.

Use Case 1: Travel Agency

Feature 1	Login To An Account
User Story	As a travel agency, I can login to an account such that I can access a dedicated dashboard of my current travellers.
Scenario	Bobs Agency will login into their account using their username and password which allows them to view their dashboard.
Example	

garrysmith@gmail.com	I need help booking a family vacation to the Caribbean.	Stanley Parks	The hotel room was not booked. I need a new room.
jamesdaniels@gmail.com	I would like to travel to Amsterdam from Manchester	Jon Jones	Could you please postpone my flight by two weeks?

Current Clients

Client Name	Email	Phone Number	Destination	Location
Stanley Parks	stanleyparks@gmail.com	083 555 6733	Bangkok, Thailand	Sydney, Australia
Edgar Wright	ewright@gmail.com	074 555 1491	Swansea, Wales	Birmingham, England
Maria de Souza	mdsouza@gmail.com	073 555 3921	Washington, USA	Toronto, Canada
Jon Jones	jjones@gmail.com	084 555 4143	Durban, South Africa	Manchester, England
Sibusiso Jacob	sjakes@gmail.com	074 555 8127	Windhoek, Namibia	Cape Town, South Africa

Feature 2	List of Travellers
User Story	As a travel agency, I can view a list of current clients with such that I can easily interact with a specific user.
Scenario	Bobs Agency will login into their account and see the email, phone number, destination and location of each of their customers such as Stanley Parks.
Example	

Back to Main Page

Dashboard of Bob's Agency

New Requests

User Email	Request
garrysmith@gmail.com	I need help booking a family vacation to the Caribbean.
jamesdaniels@gmail.com	I would like to travel to Amsterdam from Manchester

Client Issues

Client Name	Issue
Stanley Parks	The hotel room was not booked. I need a new room.
Jon Jones	Could you please postpone my flight by two weeks?

Current Clients

Client Name	Email	Phone Number	Destination	Location
Stanley Parks	stanleyparks@gmail.com	083 555 6733	Bangkok, Thailand	Sydney, Australia
Edgar Wright	ewright@gmail.com	074 555 1491	Swansea, Wales	Birmingham, England
Maria de Souza	mdsouza@gmail.com	073 555 3921	Washington, USA	Toronto, Canada
Jon Jones	jjones@gmail.com	084 555 4143	Durban, South Africa	Manchester, England
Sibusiso Jacob	sjakes@gmail.com	074 555 8127	Windhoek, Namibia	Cape Town, South Africa

Feature 3	New Client Communication
User Story	As a travel agency, I can receive travel requests from customers such that I can communicate with new clients and set arrangements or quotes.
Scenario	Bobs Agency receives a message from Garry Smith to book a family vacation. Bobs Agency can then respond via writing to their email.
Example	

Back to Main Page

Dashboard of Bob's Agency

New Requests

User Email	Request
garrysmith@gmail.com	I need help booking a family vacation to the Caribbean.

Client Issues

Client Name	Issue
Stanley Parks	The hotel room was not booked. I need a new room.

jamesdaniels@gmail.com	I would like to travel to Amsterdam from Manchester	Jon Jones	Could you please postpone my flight by two weeks?

Current Clients

Client Name	Email	Phone Number	Destination	Location
Stanley Parks	stanleyparks@gmail.com	083 555 6733	Bangkok, Thailand	Sydney, Australia
Edgar Wright	ewright@gmail.com	074 555 1491	Swansea, Wales	Birmingham, England
Maria de Souza	mdsouza@gmail.com	073 555 3921	Washington, USA	Toronto, Canada
Jon Jones	jjones@gmail.com	084 555 4143	Durban, South Africa	Manchester, England
Sibusiso Jacob	sjakes@gmail.com	074 555 8127	Windhoek, Namibia	Cape Town, South Africa

Feature 4	Current Client Communication
User Story	As a travel agency, I can receive message requests from current customers such that I can set or change arrangements that were made with the customer.
Scenario	Bobs Agency receives a message from Jon Jones to change their flight. Bobs Agency can then respond to the customer, and change the flight of the plane.
Example	

Back to Main Page

Dashboard of Bob's Agency

New Requests

User Email	Request
garrysmith@gmail.com	I need help booking a family vacation to the Caribbean.
jamesdaniels@gmail.com	I would like to travel to Amsterdam from Manchester

Client Issues

Client Name	Issue
Stanley Parks	The hotel room was not booked. I need a new room.
Jon Jones	Could you please postpone my flight by two weeks?

Current Clients

Client Name	Email	Phone Number	Destination	Location
Stanley Parks	stanleyparks@gmail.com	083 555 6733	Bangkok, Thailand	Sydney, Australia
Edgar Wright	ewright@gmail.com	074 555 1491	Swansea, Wales	Birmingham, England
Maria de Souza	mdsouza@gmail.com	073 555 3921	Washington, USA	Toronto, Canada
Jon Jones	jjones@gmail.com	084 555 4143	Durban, South Africa	Manchester, England
Sibusiso Jacob	sjakes@gmail.com	074 555 8127	Windhoek, Namibia	Cape Town, South Africa

Feature 5	Edit Traveller
User Story	As a travel agency, I can edit customer information such that the customer and I can view the updated information.
Scenario	Bobs Agency changes the destination of Stanley Parks from Bangkok to Phuket.
Example	

Current Clients

Client Name	Email	Phone Number	Destination	Location	Edit
Stanley Parks	stanleyparks@gmail.com	083 555 67331	Bangkok, Thailand	Sydney, Australia	Edit Traveller
Edgar Wright	ewright@gmail.com	074 555 1491	Swansea, Wales	Birmingham, England	Edit Traveller

Maria de Souza	mdsouza@gmail.com	073 555 3921	Washington, USA	Toronto, Canada	Edit Traveller
Jon Jones	jjones@gmail.com	084 555 4143	Durban, South Africa	Manchester, England	Edit Traveller
Sibusiso Jacob	sjakes@gmail.com	074 555 8127	Windhoek, Namibia	Cape Town, South Africa	Edit Traveller
Raymond	aaa@gmail.com	02993939908098	hihi	ooo	Edit Traveller

Editing Stanley Parks

Location

Sydney, Australia

Destination

Phuket, Thailand

Phone

083 555 67331

Submit Query

Client Name	Email	Phone Number	Destination	Location	Edit
Stanley Parks	stanleyparks@gmail.com	083 555 67331	Phuket, Thailand	Sydney, Australia	Edit Traveller
Edgar Wright	ewright@gmail.com	074 555 1491	Swansea, Wales	Birmingham, England	Edit Traveller
Maria de Souza	mdsouza@gmail.com	073 555 3921	Washington, USA	Toronto, Canada	Edit Traveller
Jon Jones	jjones@gmail.com	084 555 4143	Durban, South Africa	Manchester, England	Edit Traveller
Sibusiso Jacob	sjakes@gmail.com	074 555 8127	Windhoek, Namibia	Cape Town, South Africa	Edit Traveller

Feature 6	Add Traveller
User Story	As a travel agency, I can add a new customer from my dashboard such that I can manage information for future customers.
Scenario	Bobs Agency adds 'John Snow' as a new customer, and creates a new account for him. This account can now be accessed from 'Login as a Traveller'
Example	

Stanley Parks	stanleyparks@gmail.com	083 555 67331	Phuket, Thailand	Sydney, Australia	Edit Traveller
Edgar Wright	ewright@gmail.com	074 555 1491	Swansea, Wales	Birmingham, England	Edit Traveller
Maria de Souza	mdsouza@gmail.com	073 555 3921	Washington, USA	Toronto, Canada	Edit Traveller
Jon Jones	jjones@gmail.com	084 555 4143	Durban, South Africa	Manchester, England	Edit Traveller
Sibusiso Jacob	sjakes@gmail.com	074 555 8127	Windhoek, Namibia	Cape Town, South Africa	Edit Traveller
Raymond	aaa@gmail.com	02993939908098	hihi	ooo	Edit Traveller
Raymond	ryan@gmail.com	02939393939	South Korea	Sydney, Australia	Edit Traveller

Use Case 2: Traveller

Feature 1	Login To An Account
User Story	As a traveller, I can login to an account such that I can access a dedicated dashboard of my information.
Scenario	A traveller will login into an account using a username and password to view their dashboard.
Example	

Feature 2	Traveller Trip Information
User Story	As a traveller, I can view my trip information on my dashboard such that I can easily view all my trip information at a glance.
Scenario	A traveller will login and see the start date, end date, source and destination of their trip.
Example	

Feature 3	Traveller Disease Information
User Story	As a traveller, I can view disease information and precautions on my dashboard such that I can understand the current situation of diseases including COVID statistics.

Scenario	A traveller will login and see current reports for the location, current risk level and current COVID statistics.
Example	

Feature 4	Traveller Communication
User Story	As a traveller, I can send a message such that any problems with my trip can be answered by the corresponding travel agency.
Scenario	A traveller sends a report to their travel agency about an issue with their flight after logging into an account.
Example	

Use Case 3: Normal User

Feature 1	Specific Search
User Story	As a user, I can choose which search engine I want to use so that I can acquire the relevant information of my needs.
Scenario	A client will go to our home page and find 3 searches, with details of each feature. When the client clicks on the green button, they are sent to that specific search.
Example	

Feature 2	Search By Location
User Story	As a user, I can search by a country location such that I can quickly get a recommendation for travelling.
Scenario 1	A client will search for Australia and see that there are no reports. However due to the risk level, the user is advised to use some caution.
Example	

Scenario 2	A client will search for Malawi and see that there is a report in the last 90 days. A recommendation is made to the user, and the risk level is also shown.

Feature 3	Search By Location and Dates
User Story	As a user, I can search by a country location and dates such that I can get accurate information about reports within a certain time period.
Scenario	A traveller will search up Nigeria, and view all reports for all diseases from 2021 to 2022.
Example	

Diseases	Syndromes	Date	Locations
yellow fever, other, dengue		2021-03-19	Ghana, Nigeria, Senegal, Burkina Faso, Bono, Four
yellow fever		2021-06-19	Nigeria, Chad
yellow fever		2021-03-19	Ghana, Republic, Nigeria, Nine, Central African Republic, Niger, Congo, Chad
other		2022-03-17	Ghana, Nigeria, Senegal, Guinea, West, Niger, Central, Chad
lassa fever		2022-03-19	Centre, Nigeria
lassa fever		2022-01-02	Nigeria, Four
other, lassa fever		2022-01-30	Enugu, Ondo, Kaduna, Nigeria, Delta, Edo, Cross River
other, lassa fever		2022-05-19	Nigeria, Liberia, Sierra Leone, Guinea, West
lassa fever		2022-01-30	Nigeria
lassa fever		2022-03-19	Mali, Nigeria, Togo, Sierra Leone, Liberia, Germany

Feature 4	Search By Disease, Location and Dates
Feature	As a user, I can search by a disease, dates and location such that I get options to search for my specific needs. I will also see the links to those search results.
Scenario	A user will search for yellow fever in Nigeria, and view all reports for all diseases from 2017 to 2022.
Example	

Headline	Date of publication	Article id	url
Read Article	Yellow Fever - West and Central Africa	2021-12-23 00:00:00	https://www.who.int/emergencies/disease-outbreak-news/item/yellow-fever---west-and-central-africa
Read Article	Yellow Fever – Nigeria	2021-10-01 00:00:00	https://www.who.int/emergencies/disease-outbreak-news/item/2021-DON336
Read Article	2017 - Nigeria	2017-12-22 00:00:00	https://www.who.int/emergencies/disease-outbreak-news/item/22-december-2017-yellow-fever-nigeria-en

Feature 5	Request for A Quote
Feature	As a user, I can send a quote to all nearby travel agencies such that I can get a profession opinion on current travel options.
Scenario	A user sends a message asking for a quote on a trip to Ghana. This is shown on Bobs Agencys ‘New Requests’.
Example	

Feature 6	View Article Content
Feature	As a user, I can read an article from article search such that I do not visit an external website to view the contents of an article.
Scenario	A user clicks on Read Article from Complete Search. He can then see the article content from the linked website.
Example	

Feature 7	Copy To Clipboard
Feature	As a user, I can copy text from an article such that I can easily share information to other people.
Scenario	A user clicks on 'Copy Text' for the first paragraph. The text is then copied to the users clipboard.
Example	

Requirements of the API

When we began this project, were given the following requirements our API should be able to perform:

1. Ability to integrate data from different sources and presented in a user-friendly way
2. Ability to browse news related to a disease outbreak over a period of time/geographically, identify news about outbreak of interest
3. Ability to examine social media related posts on disease outbreaks over a period of time, identify particular trends
4. Ability to provide advanced analysis facilities such as analysing the impact of an outbreak on residents of the region over a period of time or predicting potential outbreaks based on historical patterns.

Our software fulfils these requirements in the following ways:

Feature	Solution
1	We made use of a web scraper to collect data from a source website Promed. The information is extracted into a JSON file and compiled into an easy to read report.
2	Our application collects a large amount of information from reports on Promed, the user can filter through the reports on location, date, time and disease. This allows the user to narrow down their search for a precise piece of information they are looking for.
3	We had plans for our analytics platform to be integrated with the Twitter API and examine tweets to verify disease outbreaks detected from Promed however due to time constraints we were not able to implement this feature.
4	Our API does not provide any analytics for predictions. However we do analyze risk level shown in the front end model in Use Case 1.2.

Design Details

API Model

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.

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.

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.

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.

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.

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/”.

Web Hosting

Hosting our API locally on a local server is extremely expensive and involves a lot of manual labour to start up and maintain. That is why we decided to choose an external service which will host our API instead of personal local hosting.

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>
	}
	``

Error Model	Status Code	JSON Response

Item not Found	404	```
		{
		"message":
		{
		"The item with given id was not found."
		}
		}
		``

Validation Error	422	```
		{
		"detail": [
		{
		"loc": [
		"string"
		],
		"msg": "string",
		"type": "string"
		}
		]
		}
		``

Web Scraper

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.

Web Scraper Example

Firstly, our web scraper will load a website. An example is https://www.who.int/emergencies/disease-outbreak-news/item/yellow-fever---ghana.

Then the web scraper will search for any paragraphs or HTML p classes.

After finding all paragraphs, the paragraphs are converted to JSON by finding keywords such as 'Ghana' and appending the information to a list.

```
from bs4 import BeautifulSoup as bs
import requests as rq
def getsoup(url):
    res = rq.get(url)
    soup = bs(res.content, features="lxml")
```

```
paras = []
i = 0
for link in soup.findAll('p'):
    # print(link)
```

JSON articles will store the main body of text, the headline of the original article, the url and the publication date of the article.

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.

Web Scraper Results

For our API, a web scraper is used to obtain the data, however when using our API the user will not interact with the web scraper. When searching for certain diseases, the API will query our database, and find matches on disease, location and time periods using SQL functions. This will greatly reduce waiting time for users, as searching a database will be much quicker than web scraping through articles every time an API route is called. Matching entries in the database will be formatted into JSON Objects, with each Object being the requested type of data, such as an article or report.

Sample HTTP Calls

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

Side Note

For the sample HTTP Calls, errors have not been shown. This has been intentional since it would present repetitive information about our routes. One route has been fully documented named '/api/articles/1'.

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
}
```

Development, Implementation and Deployment

API's Development

For our API, we had to make several choices on how the API was structured as well as what dependencies are needed such as libraries or external services.

To start we decided to focus firstly on what structures / systems we wanted for our API.

Database System

Since our API requires a database in order to serve information to the user, we thought about the benefits and drawbacks of NoSQL vs Relational Databases (SQL).

Feature	NoSQL	Relational Databases
Schema	Dynamic schema, this makes changes the predefined schema simple and inexpensive	Predefined, structured schema. Altering the schema can be very expensive
Data Structure	Unstructured data with no relations. Data is stored in singular tables with all information about a record stored in that table	Highly structured data stored in a predefined schema. Data that is added must match the schema
Scaling	Horizontally scalable. Inexpensive to expand with multiple low cost servers	Vertically scalable. Can prove a challenge as scaling becomes quite expensive - it requires adding/upgrading resources for your existing machine
Performance	Capable of handling even the biggest datasets	Not as performance efficient for massive datasets
Procedures/Functions	No stored procedures, difficult to identify patterns	Plenty of stored procedures to understand the data and its patterns. Functions can be made to select specific data from multiple tables.
Availability/Consistency	Databases are not always consistent, which can be confusing for the user. (Trades consistency for partition tolerance - CAP Theorem	Availability depends on the server. Data is consistent. (Trades off partition tolerance for availability and consistency - CAP Theorem
Rigidity	Data can be inserted in any form and is not necessarily in the correct format	Data is guaranteed to be in a specific format. Entries stored in predictable structure
Analytics	Better suited for real-time analytics	Better suited for analyzing past datasets

Team Experience	Our team is inexperienced in NoSQL databases, both in theory and application. Time would need to be dedicated to learning the software	Our team has considerable experience in relational databases and no time would need to be spent learning the software

Obvious drawbacks of RDBMS in comparison to NoSQL are scaling and performance. The cost of scaling can be expensive as it can only be done vertically and RDBMS performance does not scale as well as NoSQL does as the dataset grows. However, we feel these drawbacks are mitigated by the benefits they provide.

All factors considered, the datasets we will be working with are more suited to a structured, predefined schema. Our records are very predictable and will always match the schema. Analyzing past datasets is simpler, there are plenty of functions and procedures to manipulate the data to find patterns and the data is always consistent. Given all these reasons, and our team's experience with relational databases - and inexperience with NoSQL - an RDBMS is the most logical option.

Web Hosting Services

Since hosting a server locally is expensive and has heavy startup costs, we needed an external hosting service for our API. In this case we compared Infrastructure as a Service (IaaS) to Platform as a Service (PaaS). Although we are comparing different types of services, we decided to use real examples as these hosting services vary depending on the host. In our comparison we compared AWS IaaS and Heroku PaaS.

Article	Heroku	Amazon Web Services (AWS)
Service	Platform as a Service	Infrastructure as a Service
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, which is a PaaS. 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.

Programming Language

Python was chosen as a main language for our API. Although other programming languages with web hosting exist such as Ruby on Rails and JavaScript NodeJS, Python was the only programming language which everyone had experience with coding.

Python was also chosen for the reasons below:

- Python is a flexible language in the sense that it does not restrict developers from developing a certain sort of application
- Memory allocation is not required, compared to languages such as C
- It is a popular tool for data science and analytics
- There are hundreds of Python libraries to use, which are easily imported compared to languages like Ruby which does not have custom imports
- 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 and multiple issue fixes, saving time compared to lesser known like Ruby on Rails.

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.

Implementation and Software Architecture

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

- The ‘Scrapy’ web crawling framework
- The ‘FastAPI’ web framework
- The ‘PostgreSQL’ RDBMS hosted by Heroku Postgres
- Heroku, a cloud service platform
- The Linux Operating System
- Bootstrap and JS
- External API

The tech stack for the API has been visually created below along with the justification for these choices.

Justification of Software Architecture

Back End Software

Scrapy

As our API is automated, we needed a web scraper that fit our personal needs and also created in python

We compared Selenium, BeautifulSoup and Scrapy.

Article	Scrapy	Selenium	BeautifulSoup
Extensibility	Scrapy is designed for middleware to be customisable allowing for custom functionality. It is easily integrated into a data pipeline.	Selenium is the best choice when dealing with Javascript.Selenium is also beginner friendly.	Beautiful Soup helps with maintaining code simple and flexible. Ideal for low to mid level complexity projects.
Performance	Scrapy has a high performance because it is able to utilise asynchronous system calls.	Selenium is similar in performance to Beautiful Soup however is not equivalent to Scrapy.	Beautiful Soup can be quite slow when performing certain tasks however this issue can be overcome through the help of multithreading.

Ecosystem	Scrapy has a good ecosystem, it can use proxies and VPNs to automate the task and send multiple requests from multiple proxy addresses.	Selenium has a good ecosystem for development but it is unable to utilise proxies easily.	Beautiful Soup has a lot of dependencies in the ecosystem making it not ideal for a complex project.

Despite there being numerous options for web scraping, Scrapy was the better choice for our team for the following reasons:

- 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.

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

FastAPI

For our choice of web framework when it comes to developing APIs, we discussed whether to use Flask, Django or FastAPI. We chose FastAPI mainly due to the ease of use and adaptability but some other comparisons are briefly explained below:

- 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.
- It is built on the Asynchronous Server Gateway Interface (ASGI) specification, which means multiple asynchronous requests are handled easier than Flask or Django.
- is based on the open standards for APIs.
- has flexibility code wise, and doesn't restrict the code layout, unlike Django.
- automatic swagger doc creation, unlike Django or Flask

Overall Django, Flask and FastAPI in comparison have differences that are negligible in the end on a practical analysis. However FastAPI was best when it came to the overall development time it would take to produce a product.

Heroku PostgreSQL

Selecting a server to host our database is an important decision. A server poorly equipped for our application's demands will negatively impact both us as developers and the user experience. Conversely, it is possible to overestimate the demands of our application; this would lead to us only utilizing a fraction of the capabilities of our server and overpaying for what we're getting. Our server needs to be reliable, secure, scalable and cost effective.

Feature	Heroku PostgreSQL	PostgreSQL Server	PostgreSQL Server on Dedicated Server
Cost	Flexible. The lowest tier package is free, the standard package is $50/month. Package can be upgraded as required	None	Either cost of building, running and maintaining a dedicated server or cost of renting a virtual private server
Performance	Decent across all packages. Standard provides 4GB of RAM. Low downtime tolerance, high performance ceiling	Limited by local machine resources. Not reliable for external users - more suited for developing and testing. Not considered stable for the end user	Scales with cost - the more you pay the more you get. High performance ceiling
Scalability	Fully scalable as required	None, can only alter the amount of resources available to the server on the local machine	Fully scalable, can scale up with additional resources as required, can become costly
Security	Highly secure, compliant with industry standards (PIC and HIPAA). Can use own encryption key	Responsibility is on the developers to maintain the database/server's security	Responsibility is on the developers to maintain the database/server's security
Setup	Setup is fairly simple and well documented	Relatively simple to set up locally, more complicated to set up for external use	Quite challenging to set up the infrastructure
Uptime	A minimum uptime of 99.5% with the lowest package and 99.9% with the standard package	Associated with local machine's availability and network stability	Using a dedicated machine: same as Postgres Server VPS: dependant on company hosting from
Drawbacks	Hosted on Heroku so will fail if Heroku fails	Very limited in performance	Maintaining a dedicated server requires lots of time and expertise

As one might extrapolate, a PostgreSQL Server hosted on a local machine is inadequate for our application's demands - it would simply hinder us too much and result in an unstable, unreliable server for the user. Where a locally hosted server lacks the scalability and performance, a dedicated server more than makes up for. It is the most flexible option, allowing vast scalability and a high performance ceiling. The server can be customized to improve performance by altering memory management, caching, buffer pools and more. However, this comes at a cost. The amount of time required to set up and maintain this server is enormous and would require an immense amount of expertise. As our team is entirely inexperienced with network security and server maintenance, it would be illogical to choose this option. We would be out of our depth and would undoubtedly encounter more problems than avoid. It's for this reason we have gone with Heroku PostgreSQL.

The service we have access to is the same service Heroku itself uses. There are a plethora of package options and upgrading is simple, meaning we can grow our server as our application and database grows.

Additional features:

- Feature-rich: offers useful features like forking and follower instances.
- Reliable: downtime tolerance of less than 4 hours per month at the lowest package, less than 1 hour per month with the standard package and any package above is less than 15 minutes per month (99.98% uptime).
- Rollback: offers a 4 day rollback with the standard package.
- Size restrictions: standard package has no row limitations.
- Storage capacity: standard offers 64GB of storage, but the package can be upgraded to increase this.
- Performance metrics: database metrics available for analysis.

All this makes it the ideal choice for our application.

Front End Software

Bootstrap

EpidemicEagle's front end options came down to choosing from ReactJS, Bootstrap and Vue. We compared the three frameworks on a number of features for our final decision inside a table below.

Article	ReactJS	Bootstrap	Vue
Performance	Fast, lightweight Component based architecture means more robust apps, reduced code clutter, reduced DOM manipulation Component based architecture significantly makes loading pages faster, and more pleasing UX.	Criticised for slower performance Offers extensive customisation features to increase performance, despite being content heavy Workarounds and best practices can pull the time to load 1.3 MB sized pages to 2.1 seconds.	Performance can worsen with each new feature added, or each component extended. Uses lazy loading to improve runtime.
Application Architecture	No built in architecture View layer application made up of components, Render underlying UI as data changes.	Supports Model-View-Controller architecture pattern Bootstrap's role in the MVC is the view component	Uses a 'ViewModel' approach, in a Model-View-ViewModel pattern. The 'ViewModel' handles synching between Model and View parts of code. The view is the entry point to an application rather than the controller. Business logic is decoupled from UI
Ease of Testing	Quite easy to test, offers test runners E.g. Jest, Mocha test suites Reduced time to market Speeds up testing and productivity.	Needs external plugins for testing apps E.g. Chrome Developer Tools, DesignModo, BrowserStack This external testing eliminates cross browser bugs and checks app's consistency and uniformity	Uses Jest, Mocha, Chai or Vue Test Utils.
Scalability	Yes React is purely javascript traditional methods, virtual DOM and component reusability can make a React app scalable	Yes Class components, interactive grid system	No. Significant work (forking a Vue application and using multiple repositories) is required.
Suitability for Building Complex Apps	Suitable Builds highly interactive single page apps. Uses external server side rendering architecture such as Redux, Flux, Next.js	Suitable Mobile first framework means an app will be supported on various platforms, devices, screens with ease. broad range of ready-made components, templates, themes for use. 12 column grid system useful for complex web layouts across platforms.	Preferred for developing small-time applications.
User Experience	More likely to be adopted by users. High quality UI Fast rendering When built correctly, React apps deliver great performance. Frequently updated.	Responsive design Consistency across all platforms Apps made in bootstrap are always trusted and valuable because of this consistency. Can enhance UX through external UI kits.	Creates an interactive and eye-catchy UI.
Rapid Development		Quick development time due to the following:	

	Requires in depth understanding of router system, configuration specifics, and lots of Javascript experience	extensive documentation, developer friendly, doesn't require much coding knowledge, libraries consist of several class components, only requires HTML and CSS, ready made components, JS plugins, cross browser compatibility.	Quick development time due to the lightweight nature of Vue and add on tools and rendering features.
Application Size	Not a full featured frontend framework; it is simply a library. App sizes can be quite large.	Varies depending on content, need to be wary of library packages and their unused components. Following best practices can minimise the overall size.z	Vue is a lightweight framework. Application size is not heavy.
Learning Curve	If developers have JavaScript experience, then it takes little time to learn. Uses API structures and data flow in Javascript. Only versions after 16.0 are up to date.	Developer friendly, very little learning curve when prerequisites of HTML and CSS are covered.Effort might be needed in learning CSS classes and Bootstrap components. Extensive documentation exists, there is a large community of developers online for learning guides.	Easiest to learn out of the three choices, basics can be covered in less than a week. Requires basic understanding of ES6 functions, and fundamental JavaScript skills. Documentation is understandable but not rich.

There were considerable pros and cons for each but Bootstrap stood out for the following reasons:

1. Our team has chosen a rapid development time as a priority.
2. The learning curve is small for Bootstrap, and our team is not very experienced / does not have time available to spend learning and mastering JavaScript. Focusing on HTML and CSS means we can spend more time being productive, rather than worrying about learning.
3. Bootstrap still has potential to produce quality apps.
4. Our application will not require much complexity.

External APIs

To complement our own API data, we decided to combine our API with external APIs to give the user more data and information

Disease.Sh API

Disease.sh is a free to use, open disease data API. It gathers up to date information about current global outbreaks including COVID-19 and Influenza. There is a plethora of other free open-source disease data APIs to choose from however we decided to use Disease.sh as it is a widely used API and it utilises trusted sources. Another reason we use Disease.sh is because a country parameter can be used to quickly search for information regarding a specific country.

TravelAdvisory API

TravelAdvisory API is REST API designed for determining the risk level for visiting countries. It provides an automatically updating list of country advisories which contain a risk value between 0-5 describing how dangerous the country is from the traveller's perspective. Despite other APIs being available such as TravelAdvice API we decided to use TravelAdvisory API due to it having a free version.

teamViral API

We decided to adapt and combine our API with other teams API for the purpose of having multiple sources for reports and articles. There were many teams APIs to choose from however we ended up using teamViral's API because their team had a JSON dump of their data compared to other teams which required parameters for every call. This route made it easier to call their API, and get the JSON needed to then incorporate into our JSON of reports and articles.

Further Justifications

Some choices which cannot be shown in the software architecture diagram have been added below.

PyTest

For testing our API, two choices were considered: Unittest and Pytest.

	Pytest	UnitTest
Description	Popular open-source testing framework, third-party dependency	Default python testing framework that comes with the python package
Team Experience	Major	None
Test Types	Function based testing.	Method based testing.

	Tests are independent functions.	Tests are sub-methods of a defined class.
Assertion Cases	Builtin assertions	Builtin assertions
Test Results	Highlighted text with code snippets and error message	Provides line and class errors.
Test Structure	Tests are independent of each other	Tests are created inside a class.
Platforms	Python Only	Python and Java(JUnit)

Although UnitTest is an integrated feature of python, we decided to use pytest mainly due to team experience.

Summarizing other reasons why we chose of Pytest over Unittest:

- Our team has more experience with Pytest than UnitTest
- Pytest functions are independent functions instead of inside of a class
- Test errors are clearly highlighted based on function compared to UnitTest displaying a line error

Challenges addressed and shortcomings

Although our choices for our software architecture have been justified against other examples, there are also some overlying issues that need to be addressed about the tech-stack as a whole.

Heroku and Heroku PostgreSQL coupling

Our API is heavily reliant on services given by Heroku, as our web hosting and database system.

In the case that Heroku has maintenance or goes offline, our API will not be viewable to the public. Additionally all data stored inside Heroku PostgreSQL will not be accessible.

Heroku Timeouts

One of the issues with Herokus free tier for web hosting is timeouts logging during inactivity.

When our API is inactive, a fragment of error code will be logged in our log file, using up space.

When there are a lot of these error logs, then logs of previous calls to our API will be deleted and not be able to be viewed again. Additionally when our API startups from inactivity, the response time is extremely long, having a negative effect on user experience.

Application Logs

ALL PROCESSES

```
2022-03-15T08:45:56.855604+00:00 app[web.1]:     await receive()
2022-03-15T08:45:56.855605+00:00 app[web.1]:   File "/app/.heroku/python/lib/python3.9/site-packages/uvicorn/lifespan/on.py", line 135, in receive
2022-03-15T08:45:56.855605+00:00 app[web.1]:     return await self.receive_queue.get()
2022-03-15T08:45:56.855606+00:00 app[web.1]:   File "/app/.heroku/python/lib/python3.9/asyncio/queues.py", line 166, in get
2022-03-15T08:45:56.855606+00:00 app[web.1]:     await getter
2022-03-15T08:45:56.855607+00:00 app[web.1]: asyncio.exceptions.CancelledError
2022-03-15T08:45:56.855607+00:00 app[web.1]:
2022-03-15T08:45:56.856590+00:00 app[web.1]: [2022-03-15 08:45:56 +0000] [10] [INFO] Worker exiting (pid: 10)
2022-03-15T08:45:57.099206+00:00 app[web.1]: [2022-03-15 08:45:57 +0000] [4] [INFO] Shutting down: Master
2022-03-15T08:45:57.256404+00:00 heroku[web.1]: Process exited with status 0
```

☑ Autoscroll with output

Save

Shortcoming of web scraping with Dynamic Websites

Web scraping was one of the biggest challenges our team faced when developing the API. This feature was too intensive to complete and as a result, our final product does not contain a web scraper. Instead we have chosen to display preloaded data from external web scrapers.

The choice of Scrapy at first seemed to be the obvious best choice when it came to web scrapers. However the complexity of Scrapy made it difficult for our team to make use of given our short time frame to create the web scraper. In future projects, BeautifulSoup might be a better choice due to it being more beginner friendly. Another challenge we faced creating the web scraper was due to the way the website Promed loaded it's data. Attempting to scrape ProMed revealed that the website dynamically loads its data, due to heavily being based on JavaScript. That meant that we could not just simply parse HTML classes to extract data for our API, due to ProMed requiring human interaction to load the data. One solution that was attempted was to send a post response with a body and parse data from that URL, however this was abandoned due to time issues.

Shortcoming of PostgreSQL

Since our team had prior experience in database implementation in PostgreSQL, our database schema was constructed in such a way that made it very efficient for PostgreSQL queries. However this was difficult to implement in Python due to the standardized pythonic library of SQLAlchemy. FastAPI only had documentation for SQLAlchemy, compared to insignificant documentation for PostGreSQL.

An option considered by our team was to change our schema to fit SQLAlchemy, given our knowledge of the language. However this change would make an extremely inefficient schema for a relational database, and would contradict all our previous database justifications. Ultimately, it became too complicated to implement a database effectively and as a result we decided to use a JSON file to store our data instead of a relational database. This solution allows our website to access and store data, however it also provides an unsafe method of storage, being susceptible to attacks or JSON leaks.

Key Achievements and Benefits of Our Software

Heroku Hosting

Heroku was difficult to set up to link to GitHub, This was mainly due to our project being inside of a sub directory, when Heroku normally uses the main directory. This was achieved by setting a PATH variable in Heroku, and using an external build pack to access the server file. This way GitHub could be linked with Heroku without any changes to the GitHub structure. Additionally Heroku could detect any changes in the branch, so automatic deployment was guaranteed with a successful pipeline. This is a

Front End

Our front end provides a unique experience for any user that visits. It calls the back end for data, and returns the information in a visual way, using aids such as tables. Due to external APIs, one benefit of our software is the combination of several sources of information into a uniform way to easilly share to our users. Copy-To-Clipboard function was also a quick but unique benefit of our website that not many websites implement as well.

Back End

Using FastAPI, our back end code allows the hosting of a RESTful API as well as a static website all in one server file. This was a key achievement for our software, since our website does not require two separate website hosts in order to share our software.

Team Structure

Our team was composed of mainly developers. This is reflected in the main roles delegated for each person. Additionally, our team planned to delegate the more important roles to the more experienced people in our team.

Name	Main Roles	Reasons	Responsibilities
Aaron	Scrum Master, Full Stack Developer, Team Leader, QA Engineer	Highly experienced coder with experience with HTML, CSS, Python. Has experience in prior job with RESTful API's. Has great interest in managing the front end and back end of the project.	Develop the initial back end and front end of the software. Test the final product by manual and automated user experiences.
Sandy (departed)	Project Manager, Developer	High interest in the project, including development and back end functionality	Initialize and layout the back end element of the project. This includes the database design as well as the web scraper design.
Treav (departed)	Back-end Developer, Tester	High interest in the development of new software, especially in the technical areas.	Support team members in areas such as documentation, testing and coding.
Mihail	Back-end Developer	High interest in the development of new software, especially in the technical areas.	Support team members in areas such as documentation, testing and coding.
Steyn	Product Owner, Back-end Developer	Knowledgeable of higher end database systems. Past history with business propositions. Has connections with target audience of the software.	Develop the business idea, and reflect the features needed to the Scrum Master. Work on the database setup, with the Project Manager.

Software Tools

In order for our team to create and schedule tasks to do, external software was used to handle the management of the team. Our team used the following software tools to manage the development and production of the software.

Jira

Jira is a software solution we decided to use to assign tasks and set deadlines for our project. Before comparing Jira to other alternatives, Jira was highly favored by all members of the team due to experience. Bitbucket was an option for our team, which would allow for code review and Docker repository support unlike Jira, however since only some of the team had experience with the software, and Jira was ultimately chosen for our project management software.

For this software project, we decided to establish a 10 day work period for each deliverable. This would allow team members to plan ahead and maximize efficiency working on the tasks set for each person. A 7 day rest period was also set after every deliverable, to allow team members enough time to complete outside tasks such as other subjects and work employment.

Confluence

Confluence is a new software application to all members in the group. However, the tool has built a reputation to be at the forefront when it comes to visual information and collaborative efforts.

Its key features focus on structure, the idea of ‘pages’ that exist in nested hierarchies called ‘page trees' to make finding work quicker, and work spaces to keep your teams organized.

Compared to Google Docs, Confluence offers a unique and visual way of demonstrating our software project, which has shortcuts for formatting items such as images or tables. A drawback of Confluence is that it does not provide any real-time collaboration support, possibly resulting in future issues with overwriting. However we believe that Confluence overachieves over its negative effects, and was chosen as our report environment.

GitHub

GitHub is a popular tool, commonly used to share and distribute software. It has multiple features as merge requests, and commit history which acts to advantage a team environment.

Unlike BitBucket, GitHub provides GitHub Actions, which is a proprietary tool which allows add-ons such as pipelines. This would allow any commits to be cross checked with a pytest before hosting the code on the web server. Although Bitbucket has extra add-ons which emulate this feature, the team had more experience with GitHub UI compared to BitBucket. However Bitbucket would be an overall better solution for cross-site integration with Confluence and Jira, such as repository errors shown on Jira, stemming from BitBucket.

Due to the teams lack of experience with BitBucket, GitHub was our final solution for sharing and collaborating with our software.

Discord

Discord is an instant messaging / voice communication platform. Our group used it as our primary place for general communication from our phones or laptops whether we are at home or on the go. The service allows calling, video chat, texting, from either our laptops, phones or PCs, and is overall just a very easy software solution for out team to keep in contact.

Availability Tool: “When2meet”

‘http://when2meet.com

’ is a website that our team uses for everyone to jot down what times they are available or unavailable so that we can find times to schedule team meetings. Currently we have scheduled our team meetings during our Tuesday mentoring sessions, and Sunday mornings.

1:00 AM
2:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM
8:00 AM
9:00 AM
10:00 AM
11:00 AM
12:00 PM
1:00 PM
2:00 PM
3:00 PM
4:00 PM
5:00 PM
6:00 PM

Google Docs

Currently we intend to use Confluence as our central documentation tool. However, Confluence is also a new technology that most of our team is still learning. For our initial meetings and notes, we have been using Google Docs, a free online document editor. The only reason we still used Google Docs, is because everybody in our team has been using it as a reliable solution in the past few years. Even with the intention to learn and migrate to Confluence, Google Docs still remains useful when needing to take quick notes for a meeting, or jot down quick raw thoughts.

Microsoft Teams

Microsoft Teams is our official group communication platform, unlike our informal discord. Here we meet weekly with our mentor for Q&As, video calls, and presentation in a business like manner. This allows for quick communication to ask any emergency questions or have a long discussion with our mentor.

Deliverable Workload

The workload for each deliverable has been detailed below.

Deliverable 1

Name	Design Details	Management Information
Aaron	Context API Model Design Details Sample HTTP Calls API Deployment	
Sandy (departed)	Development, Implementation and Deployment API Deployment Justification of Web Scraper	Project Planner Jira Setup Role Assignee Justification of Jira, Discord, Teams
Treav	Proof Reading	Proof Reading
Mihail	Result Gathering	Proof Reading
Steyn	SQL vs NoSQL	

Deliverable 2

Name	Repository	Report
Aaron	GitHub Pipeline Setup API Documentation API Implementation	API Testing screenshots API design details Fast API justification

	PyTest Setup Heroku build checker API Testing	Further justifications Challenges addressed and shortcomings
Sandy (departed)	Heroku Setup Heroku PostgreSQL Setup Web scraper initiation	Jira Update Role Assignee Front end Software justification
Treav (departed)	API Testing	Software Architecture Diagram API Testing
Mihail		API Testing
Steyn	PostgreSQL research and setup	Database System justification

Deliverable 3

Name	Front end	Back end	Demo
Aaron	HTML file setup HTML Style	HTML implementation Route functionality HTML Style GitHub Repository updating	Presenter
Sandy (departed)		Selenium Web Scraper	
Treav (departed)		Copy Paste Function	Presenter
Mihail	HTML Style		Presenter, Script, Slides
Steyn		Fast API SQL functions to link to PostgreSQL server	Presenter, Script

Deliverable 4

Name	Repository	Report
Aaron	Login Screen Add and Edit Traveller Traveller Dashboard Travel Agency Dashboard External API integration	Management Information Introduction Our Solution Use Cases and Requirements Use Case Screenshots Web Scraper Example
Sandy (departed)		
Treav (departed)	Traveller Edit Traveller Dashboard	
Mihail	Login Screen	Our Solution Overview of Features Requirements of API External APIs justification Shortcoming of Webscraper
Steyn	Travel Agency Dashboard	Conclusion Our Solution

		Shortcoming of PostgreSQL

Reflection

Imbalanced Tasks

Our team was not very successful in finishing the tasks set at every deliverable. Many of our teammates were not experienced with current software, and such most of the tasks delegated were not done by the deadline. The team members that were familiar with the software had to work on these tasks and as a result only two of the 5 people team could work on the majority of the software. This led to workload imbalance, and as a result led to an overall sub-par software solution. In the future an initial discussion may be needed to delegate tasks depending on the skill level of each developer.

Departed Member

Due to personal reasons, a team member left the team. This was proven to be a major problem, because they had worked on an integral part of the software, and could not be easily replaced since most tasks that were delegated were singular tasks. As a result some of the software was left uncompleted for the final product. In the future, multi-delegation is required to ensure features of the product are not reliant on a singular party.

Jira Task Delegation

Jira was planned to be our frequently used management software for task delegation. However in practice, Jira was not used due to the complexity involved of adding, removing or editing tasks every deliverable, and Discord was used to detail the main tasks. This issue occurred because Jira was known by all teammates, however not enjoyed to use or to commit to. In the future, a better management software could have been discussed and chosen for easier use and task delegation.

Teamwork

Some of the major problems involved in this project was team motivation and team communication. Most of the team were unmotivated to finish tasks or write up minor details in the report. This could have been fixed by delegating tasks to multiple people with resource sharing and Q&A's rather than self-learning. Team communication was also another glaring issue with the team, where only one teammate was asking for help. This lead to undetected flaws with our software, which was unable to be fixed before the deadline. Frequent checkups could have helped resolve these unseen issues within the team before hard deadlines, reducing stress of the rest of the team.

Future Practices

Some future practices to consider for future projects are:

- Consistent checkups with non-responsive teammates
- Peer-programming and resource sharing between teammates
- Long discussion for initial software choices, especially management software
- Long discussion of skill level, including prior jobs or front end capabilities

Conclusion

In conclusion, we feel we developed a high quality and unique solution, but more importantly, we as a team were able to push through many challenges. Throughout the term, with the aid of tutor feedback, our problem space and target audience evolved to improve the uniqueness of our solution in the market. We started off very generally, and as the term progressed, we were able to find our niche and a solution which both addresses the problem and offers potential as a business.

As is well documented in Management Info, we lost two members throughout the term at integral parts of the project. This was very difficult to adjust to and really amplified the remaining members' workloads. The departing of one member in particular caused significant issues as he had been solely entrusted with a critical section of the project. In future team projects, it would be beneficial to handle the distribution of work using Segregation of Duties (SOD), whereby the responsibilities of a key part of the project are shared by two or more members, so as to mitigate the impact of departing members and to hopefully improve the quality of the solution via constant collaboration.

On the topic of workload, there was a slight imbalance in the work completed by members. This was the result of some members being slower to finish assigned tasks than others. Whilst understandable considering the difficulty associated with balancing a student's commitments, it did impact the final product. In future endeavors, it would be wise to have a lengthy discussion before the commencement of the project to establish the ability of each member - including their specialties and limitations. As a large portion of the project were concepts unfamiliar to us, it would be also beneficial to undertake the learning of new concepts together as a team, as opposed to all learning concepts separately. This would accelerate the process and allow us to collaborate more effectively.

In retrospect, when considering the challenges faced and the condensed timeline, the overall project was successful. We have come out of it with a better understanding of team dynamics. The only downfall in our project was due to task delegation and team motivation. In future projects, we as a team have learnt to communicate and work more effectively.