

SENG3011 20T1

Final Project Report

# APInteresting

*Cher Ping Choy z5135745*

*Wenlue Zhang z5158333*

*Linchen Chen z5163479*

*Ryan Cai z5210636*

*Yang Zheng z5210646*

<b>1. Executive Summary</b>	<b>1</b>
1.1 Background	1
1.2 Project Description	1
1.3 Target Users	1
<b>2. Requirements</b>	<b>2</b>
2.1 Must-have	2
2.2 Should-have	2
2.3 Could-have	2
2.4 API Requirement	2
<b>3. Use-Cases</b>	<b>4</b>
3.1 Must-have	4
3.1.1 Requirement 1	4
3.1.2 Requirement 2	5
3.1.3 Requirement 3	7
3.1.4 Requirement 4	9
3.2 Should-have	10
3.2.1 Requirement 5	10
3.2.2 Requirement 6	12
3.2.3 Requirement 7	14
3.3 Could-have	15
3.3.1 Requirement 8	15
3.3 API Use Case	16
3.3.1 Requirement 11	16
3.3.2 Requirement 12	17
3.3.3 Requirement 13	17
3.3.4 Requirement 14	18
<b>4. System Design</b>	<b>19</b>
4.1 Backend Design	19
4.1.1 API Protocol	19
4.1.2 Independence	19
4.1.3 Versioning	19
4.1.4 RESTful	19
4.1.5 Deployment	19
4.1.6 Testing	20
4.2 Front-end Design	20
4.2.1 Modularization	20
4.2.2 Data-Driven	20
4.3.3 UI Library	20
4.3 Software Architecture	21
4.3.1 Backend Structure	21

4.3.1.1 API Structure	21
4.3.1.2 Scraper Structure	22
4.3.1.2 Nature Language Processor (NLP) Structure	22
4.3.2 Front-end Structure	23
4.3.2.1 Statistics Component Structure	23
4.3.2.2 Social Impact Component Structure	24
4.3.2.2 Articles Component Structure	24
4.3.2.2 General Health Care Component Structure	25
<b>5. Technology Stack</b>	<b>26</b>
5.1 API implementation Language	26
5.1.1 Final choice	26
5.1.2 Justification	26
5.1.3 Encountered Challenges	26
5.1.4 Limitations	26
5.2 Database Component	26
5.2.1 Final choice	26
5.2.2 Justification	26
5.2.3 Encountered Challenges	27
5.2.4 Limitation	27
5.3 Text Mining Component	27
5.3.1 Final Choice	27
5.3.2 Justification	27
5.3.3 Encountered Challenges	28
5.3.4 Limitation	28
5.4 Scraper Component	29
5.4.1 Final Choice	29
5.4.2 Justification	29
5.4.3 Encountered challenges	29
5.4.4 Limitation	30
5.5 Geocode Component	30
5.5.1 Final Choice	30
5.5.2 Justification	30
5.5.3 Encountered Challenges	30
5.5.4 Limitation	31
5.6 Operating System	31
5.7 Server Application	31
5.7.1 Final Choice	31
5.7.2 Justification	31
5.7.3 Limitation	31
5.8 Front-end Framework	31
5.8.1 Final Choice	31

5.8.2 Justification	32
5.8.3 Limitation	32
5.9 Front-end Library	32
5.9.1 Final Choice	32
5.9.2 Justification	32
5.9.3 Limitation	32
<b>6. System Implementation</b>	<b>33</b>
6.1 Backend Implementation	33
6.1.1 Client Identification	33
6.1.2 Basic Flow	33
6.1.3 Data Processing Flow	34
6.1.4 Data Storage	35
6.1.5 Logging	35
6.1.6 API Usage	36
6.1.7 Library Usage	36
6.2 Front-end Implementation	36
6.2.1 User Interface	36
6.2.2 Data Visualisation	36
6.2.3 API Usage	36
6.2.3 Interface Walkthrough (Screenshots)	37
<b>7. Testing Summary</b>	<b>45</b>
7.1 Testing Process Summary	45
7.2.1 API & Database Component	45
7.2.1.1 Environment	45
7.2.1.2 Process Summary	45
7.2.1.3 Overview of Test Cases	45
7.2.1.4 Details of Test Cases	47
7.2.1.5 Coverage	47
7.2.1.6 Limitations	47
7.2.1.7 Improvement	48
7.2.2 Scraper Component	48
7.2.2.1 Environment	48
7.2.2.2 Overview of Test Cases	48
7.2.2.3 Details of Test Cases	49
7.2.2.4 Coverage	50
7.2.2.5 Limitations	50
7.2.2.6 Improvement	50
7.2.3 Data Processing (NLP) Component	50
7.2.3.1 Environment	50
7.2.3.2 Overview of Test Cases	50
7.2.3.3 Coverage	51

7.2.3.4 Details of Test Cases	51
7.2.3.5 Limitations	53
7.2.3.6 Improvement	53
<b>8. Group &amp; Project Management</b>	<b>54</b>
8.1 Work Distribution	54
8.2 Grantt Chart	55
8.3 WBS Diagrams	55
<b>9. Evaluations</b>	<b>56</b>
9.1 Requirements Status Table	56
9.2 Key Achievements Summary	57
9.3 Wished Prerequisites Skills	57
9.4 Challenges and Decisions	58
9.5 Further Improvements	59

# 1. Executive Summary

## 1.1 Background

COVID-19 spreading, as a global health crisis, has an incredibly negative impact on human's life. The project has been developed in collaboration with the Integrated Systems for Epidemic Response (ISER), which is an NHMRC Centre for Research Excellence located at UNSW. ISER monitors global epidemics and provides critical analysis of outbreaks of public health significance. ISER also conducts applied systems research, enhances collaboration and builds capacity in health systems research for epidemic control. It brings together experts in field epidemiology and epidemic response, military experts, international law and risk science experts, and government and non-government agencies involved in the epidemic response. The project description is meant to contribute towards automating and epidemics detection system which uses openly available data sources.

## 1.2 Project Description

The project is designed around developing a system that is able to collect, transform and visualise variety data in different aspects and from different websites in a reasonable way, which is also the core purpose of the system. Therefore, the system is able to scrape, handle, and store articles and data from multiple authoritative websites. The system also provides user access to data that could possibly be used to measure the social impact of diseases in an integrated view and may be used for disease monitoring, with the functionality of exporting into different formats. Furthermore, the system is able to provide general knowledge and news about diseases to the public in order to incorporate the effort made on reducing disease epidemics.

The development process has been divided into 2 phases. In phase 1, we generated disease outbreak data from one particular website, analyzed it and built our own API endpoint with it then deployed API on a cloud server. In phase 2, we use a series of APIs to build a web page in order to provide pragmatic features related to data visualization.

## 1.3 Target Users

- Data scientists and analysts
- Medical officers
- Academic medical researchers
- Public people who interested in diseases

## 2. Requirements

### 2.1 Must-have

1. As a medical officer, I want to be able to learn about specific diseases from articles on different websites so that I can know how severe the outbreak is and stay informed.
2. As a data scientist, I want to be able to know the location for different disease outbreaks over a period of time so that I can analyse the severity of different areas.
3. As a medical officer, I want to be able to monitor the severity of outbreaks on statistics in different places in the world so that I can adjust the policy to protect the public.
4. As a data analyst, I want to be able to obtain some meta-information for an article without actually reading the content so that it is easier for me to handle the article data.

### 2.2 Should-have

5. As a data scientist, I want to be able to see what the public is searching for an outbreak.
6. As a data scientist, I want to be able to see how social media reacts to a disease outbreak.
7. As a public, I want to be able to find out advice about how I can protect myself from a disease.

### 2.3 Could-have

8. As a data scientist, I want to be able to see if there is a relationship between the stock market and an outbreak.
9. As a data scientist, I want to be able to see if there is a relationship between the currencies and an outbreak.
10. As a data scientist, I want to be able to see if there is a relationship between the consumer price and an outbreak.

### 2.4 API Requirement

11. As a user of API, I want to get a list of article overviews over specific criteria so that I can demonstrate a preview of them in my application.
12. As a user of API, I want to get a list of articles in detail over specific restrictions so that I can display them on my application.
13. As a user of API, I want to get an article in detail by ID so that I can display them in my application.

14. As a user of API, I want to get analyzed reports from a specific article so that I can perform an analysis on it.



## 3. Use-Cases

### 3.1 Must-have

#### 3.1.1 Requirement 1

<b>Use Case 1</b>	Filter articles
<b>Priority</b>	Must-have
<b>Requirement 1</b>	User can learn about specific diseases from articles on different websites.
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User can search for articles
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"><li>1. The user can see a list of articles</li><li>2. The user can alter the list base on the filter provided by date interval/location/keywords</li><li>3. Articles in the list should have a title, date of publishing, and part of the main text.</li><li>4. When multiple filters are set, the filtering effects should be composited.</li><li>5. When there is no article satisfying a specific set of filtering criteria or the criteria are invalid, the list should be substituted by a notice.</li><li>6. There should be a button enabling the user to view the original page of the article.</li></ol>

<b>Use Case 2</b>	Search & Read
<b>Priority</b>	Must-have
<b>Requirement 1</b>	User can learn about specific diseases from articles on different websites.
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User can search for articles and read them
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"><li>1. User should be able to click the search button to search articles</li><li>2. There should be a button enabling the user to view the original page of the article.</li></ol>

	3. There should be a button enabling the user to view the te contained reports
--	--

<b>Use Case 3</b>	View Articles Storage
<b>Priority</b>	Must-have
<b>Requirement 1</b>	User can learn about specific diseases from articles on different websites.
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User can view the number of articles under a different category in the storage.
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view a page contain storage information</li> <li>2. User should be able to obtain a number of articles of different category by hover mouse on the chart</li> <li>3. Use should see all categories on the chart</li> <li>4. The chart of storage should be able to export for further study</li> </ol>

### 3.1.2 Requirement 2

<b>Use Case 1</b>	Filter & Search Outbreak
<b>Priority</b>	Must-have
<b>Requirement 2</b>	User should be able to know the location for different disease outbreak over a period of time
<b>Actor</b>	Data scientist
<b>Use Case Overview</b>	User can search outbreak over the different country or on a global scale within a period of time
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to specify "Global" as country name to search for global outbreaks</li> <li>2. User should be able to specify the name of a country to search for that country</li> <li>3. User should be able to type in start time and end time manually</li> <li>4. User should be able to specify a period of time quickly by clicking tags</li> <li>5. User should be able to click a button to perform a search</li> </ol>

<b>Use Case 2</b>	View Global Outbreak
<b>Priority</b>	Must-have
<b>Requirement 2</b>	User should be able to know the location for different disease outbreak over a period of time
<b>Actor</b>	Data scientist
<b>Use Case Overview</b>	User can view outbreak locations on a global scale
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to select categories of outbreaks to display by checkbox</li> <li>2. User should be able to view different categories of outbreaks in different coloured labels</li> <li>3. User should be able to see selected outbreak location on the world map</li> </ol>

<b>Use Case 3</b>	View Particular Country's Outbreak
<b>Priority</b>	Must-have
<b>Requirement 2</b>	User should be able to know the location for different disease outbreak over a period of time
<b>Actor</b>	Data scientist
<b>Use Case Overview</b>	User can view a country's outbreak locations
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to select categories of outbreaks to display by checkbox</li> <li>2. User should be able to view different categories of outbreaks in different coloured labels</li> <li>3. User should be able to see selected outbreak location on the map</li> <li>4. The centre of the map should be the specified country</li> </ol>

### 3.1.3 Requirement 3

<b>Use Case 1</b>	Filter & Search
<b>Priority</b>	Must-have
<b>Requirement 2</b>	User should be able to know the location for different disease outbreak over a period of time
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User can search outbreak over different countries or on a global scale within a period of time
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"><li>1. User should be able to specify “Global” as country name to search for global outbreaks</li><li>2. User should be able to specify the name of a country to search for that country</li><li>3. User should be able to type in start time and end time manually</li><li>4. User should be able to specify a period of time quickly by clicking tags</li><li>5. User should be able to click the button to perform a search</li></ol>

<b>Use Case 2</b>	Outbreak Overview
<b>Priority</b>	Must-have
<b>Requirement 3</b>	User should be able to monitor the severity of outbreaks on statistics in different places in the world
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User should be able to observe overview information from a chart
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"><li>1. User should be able to view all relevant categories of outbreaks in a chart</li><li>2. User should be able to view the number of outbreak cases of each category on the chart</li><li>3. Different category of outbreaks should be shown in different colours on the chart</li><li>4. The chart should be able to accept both global-scale data or single-country data</li><li>5. The title of the chart should display the name of the specified location</li><li>6. The chart can be exported for future study</li></ol>

<b>Use Case 3</b>	Outbreak Accumulated Outbreak Graph
<b>Priority</b>	Must-have
<b>Requirement 3</b>	User should be able to monitor the severity of outbreaks on statistics in different places in the world
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User should be to observe a graph that depicted the accumulated cases of outbreaks over time
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to choose which categories of outbreaks can be displayed on the graph</li> <li>2. User should be able to choose to display the total number of outbreaks on the graph</li> <li>3. User should be able to obtain the relationship of accumulated cases of an outbreak versus time from the graph.</li> <li>4. The graph should able to display multiple categories of outbreaks and labels. Each of them should be in a unique colour</li> <li>5. User should be able to see the exact number of cases of an outbreak at selected day by hover the mouse</li> <li>6. The Graph should be able to be exported</li> </ol>

<b>Use Case 4</b>	Outbreak New Outbreak Graph
<b>Priority</b>	Must-have
<b>Requirement 3</b>	User should be able to monitor the severity of outbreaks on statistics in different places in the world
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User should be to observe a graph that depicted the new cases of outbreaks over time
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to choose which categories of outbreaks can be displayed on the graph</li> <li>2. User should be able to choose to display the total number of outbreaks on the graph</li> <li>3. User should be able to obtain the relationship of new cases of an outbreak versus time from the graph</li> <li>4. The graph should able to display multiple categories of outbreaks and label each of them in a different colour</li> </ol>

	5. User should be able to see the exact number of cases of an outbreak at selected day by hover the mouse 6. The Graph should be able to be exported
--	---

<b>Use Case 5</b>	View COVID-19 Data
<b>Priority</b>	Must-have
<b>Requirement 3</b>	User should be able to monitor the severity of outbreaks on statistics in different places in the world
<b>Actor</b>	Medical Officer
<b>Use Case Overview</b>	User should be to view the latest data of reported cases of COVID-19 over different country
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to see the latest data of confirmed, suspected, cured, death cases of COVID-19 of different countries</li> <li>2. All countries should be displayed in reversed sorted order</li> <li>3. The data should be able to be exported in JSON or CSV as per selected</li> </ol>

### 3.1.4 Requirement 4

<b>Use Case 1</b>	View report of the article
<b>Priority</b>	Must-have
<b>Requirement 4</b>	User can see meta-information for an article without knowing the main text of the article.
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view the report part of an article
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. The user can see a “view report” button for every article in the list.</li> <li>2. When the user clicked the button, the report generated from the article should pop up.</li> <li>3. If there are multiple reports, all should be displayed.</li> <li>4. When the user selects a specific report, the user should be able to see the report containing the event date, mentioned locations, mentioned diseases and mentioned symptoms if these data is available.</li> </ol>

<b>Use Case 2</b>	View the location in Google Map for a location of the report
<b>Priority</b>	Must-have
<b>Requirement 4</b>	User can be navigated to Google Map displaying the location mentioned in the report
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view the location of a mentioned location in Google Map
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. In the report pop up, there should be a google map link next to every location name.</li> <li>2. When any link is clicked, the user should be navigated to Google Map centred at the location.</li> </ol>

## 3.2 Should-have

### 3.2.1 Requirement 5

<b>Use Case 1</b>	Popularity Of Google Search Keywords
<b>Priority</b>	Should-have
<b>Requirement 5</b>	User should be able to see what the public is searching for an outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view the popularity of search keywords form google
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view a list of disease-related keywords and their corresponding popularity</li> <li>2. The keywords should be reversed sorted</li> <li>3. User should be able to view more keywords by switching pages</li> <li>4. User should be able to navigate to a particular search page by clicking keyword entry</li> <li>5. Use should be able to view the increasing rate of popularity of each keyword after switching conditions</li> </ol>

<b>Use Case 2</b>	COVID-19 Search Interest by time
<b>Priority</b>	Should-have
<b>Requirement 5</b>	User should be able to see what the public is searching for an outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view the search interest of COVID-19 related keywords over a period of time on graph
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view the change of search interest of keywords on the chart</li> <li>2. The chart should be able to reflect all keyword</li> <li>3. Different keywords should be marked in different colour</li> <li>4. Use should be able to view exact popularity of a keyword at a time by mouse hovering</li> </ol>

<b>Use Case 3</b>	COVID-19 Search Frequency by location
<b>Priority</b>	Should-have
<b>Requirement 5</b>	User should be able to see what the public is searching for an outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view the search interest of COVID-19 related keywords over different states of australia
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to see states of Australia by map or by list</li> <li>2. User should be able to see search frequency in different state by mouse hovering</li> <li>3. The magnitude of frequency should be linked to intense of colour (white to yellow) on map</li> <li>4. The use should be able to select to display by map or display by list</li> </ol>



<b>Use Case 4</b>	COVID-19 Trending Question
<b>Priority</b>	Should-have
<b>Requirement 5</b>	User should be able to see what the public is searching for an outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to obtain the trending questions related COVID-19
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view a list of trending questions</li> <li>2. Questions should be reversed sorted by popularity</li> <li>3. User should be able to click the question entry to navigate to the corresponding search page</li> </ol>

### 3.2.2 Requirement 6

<b>Use Case 1</b>	Twitter Top Popular Hashtag
<b>Priority</b>	Should-have
<b>Requirement 6</b>	User should be able to see how social media reacts to a disease outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view information of top 10 popular hashtags related to disease
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to see a list of hashtags and their corresponding value at name, popularity, correlation degree with the disease, weekly trend and monthly trend on a table</li> <li>2. User should be able to click the tag to sort all entries by that value</li> <li>3. User should be able to click the tag again to reverse sort by that value</li> <li>4. Each hashtag entry should contain a graph to demonstrate the historical popularity trend of that hashtag</li> <li>5. The graph should enable mouse hover function to display exact popularity value at a specific time</li> <li>6. The weekly trend/monthly trend should use both</li> </ol>

	numbers and labels to show the trend of the hashtag (going up or going down).
--	---

<b>Use Case 2</b>	Country's Favourite Hashtag
<b>Priority</b>	Should-have
<b>Requirement 6</b>	User should be able to see how social media reacts to a disease outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view popular hashtags related to disease over different country
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to specify a country or global and search for popular disease hashtags</li> <li>2. User should be able to see labels of different hashtags with corresponding tweet volume</li> <li>3. User should be able to click the side tag to filter the hashtags</li> <li>4. All hashtag entries should be clickable and navigate the user to the specific twitter page</li> </ol>

<b>Use Case 3</b>	Disease Hashtag Historical Trend
<b>Priority</b>	Should-have
<b>Requirement 6</b>	User should be able to see how social media reacts to a disease outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view historical trend patterns of a fix list of twitter disease-related hashtags
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. A chart should evaluate the historical trend patterns of a list of diseases over time for the user</li> <li>2. User should be able to select which disease to be displayed on the chart by ticking the checkbox</li> <li>3. The chart should be able to display multiple disease hashtag</li> <li>4. The chart should be able to be exported as PNG</li> </ol>

<b>Use Case 4</b>	Latest Tweets Feed
<b>Priority</b>	Should-have
<b>Requirement 6</b>	User should be able to see how social media reacts to a disease outbreak
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view a list of latest tweets feed related to disease
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view a list of tweets</li> <li>2. Each tweet should display their names, retweeted times and content</li> </ol>

### 3.2.3 Requirement 7

<b>Use Case 1</b>	Health Care Tips
<b>Priority</b>	Should-have
<b>Requirement 7</b>	User should be able to find out advice about how I can protect myself from a disease
<b>Actor</b>	Public people
<b>Use Case Overview</b>	User should be able to view information about health care tips
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view a list of health care tips at the allocated page</li> <li>2. Health care tips should be the latest generate data</li> <li>3. User should be able to view title and a part of the main content</li> <li>4. User should be navigated to the source page when they click the entry</li> </ol>

<b>Use Case 2</b>	Advice For Protecting
<b>Priority</b>	Should-have
<b>Requirement 7</b>	User should be able to find out advice about how I can protect myself from a disease
<b>Actor</b>	Public people
<b>Use Case Overview</b>	User should be able to view advice about how to keep safe during the pandemic
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view a list of advice</li> <li>2. The length of each advice should be kept in short</li> </ol>

### 3.3 Could-have

#### 3.3.1 Requirement 8

<b>Use Case 1</b>	Confirmed Cases
<b>Priority</b>	Could-have
<b>Requirement 8</b>	User should be able to see if there is a relationship between the stock market and an outbreak.
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view global confirmed cases of the disease on the graph
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view global confirmed cases of the disease, over time on the graph</li> <li>2. User should be able to hover the mouse over to view the exact number of confirmed cases on that day</li> <li>3. The colour of the curve should be distinctive on the graph</li> </ol>

<b>Use Case 2</b>	Stock Market
<b>Priority</b>	Could-have
<b>Requirement 8</b>	User should be able to see if there is a relationship between the stock market and an outbreak.
<b>Actor</b>	Data Scientist
<b>Use Case Overview</b>	User should be able to view the stock market over time on the graph
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should be able to view S&amp;P 500, Nasdaq, S&amp;P/ASX 200, SSE Composite and ALL ORDINARIES over time on the graph</li> <li>2. User should be able to hover the mouse over to view exact value at that day</li> <li>3. The colour of the curve should be distinctive on the graph</li> <li>4. Each of them should be in a form that is comparable to confirmed cases</li> <li>5. User should be able to zoom in or out the graph for details</li> </ol>

### 3.3 API Use Case

#### 3.3.1 Requirement 11

<b>Use Case 1</b>	Get a list of articles in overview over specific restrictions
<b>Priority</b>	Should-have
<b>Requirement 11</b>	User can get a list of articles in overview over specific restrictions
<b>Actor</b>	API User
<b>Use Case Overview</b>	User could get a list of articles in the overview by keyword filtering
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"> <li>1. User should input the start date and end date as the required filter</li> <li>2. User could either input location and keyword as a filter or not, then send a request to the API server</li> <li>3. A JSON response will be return, which includes a list of articles in overview form regarding the filter</li> </ol>

### 3.3.2 Requirement 12

<b>Use Case 1</b>	Get a list of articles in detail over specific restrictions
<b>Priority</b>	Must-have
<b>Requirement 12</b>	User can get a list of articles in detail over specific restrictions
<b>Actor</b>	API User
<b>Use Case Overview</b>	User could get a list of articles in detail by keyword filtering
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>4. User should input the start date and end date as the required filter</li><li>5. User could either input location and keyword as a filter or not, then send a request to the API server</li><li>6. A JSON response will be return, which includes a list of articles in the detailed form regarding the filter</li></ul>

### 3.3.3 Requirement 13

<b>Use Case 1</b>	Get an article in detail by ID
<b>Priority</b>	Should-have
<b>Requirement 13</b>	User can get an article in detail by ID
<b>Actor</b>	API User
<b>Use Case Overview</b>	User could get a detailed article by article ID
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>7. User input the ID of the article and send a request to our API server</li><li>8. A JSON response will be returned and include detailed information about the particular article</li></ul>

### 3.3.4 Requirement 14

<b>Use Case 1</b>	Analyzed reports from a specific article
<b>Priority</b>	Must-have
<b>Requirement 14</b>	User can analyzed reports from a specific article
<b>Actor</b>	API User
<b>Use Case Overview</b>	User can use our API to analyze a specific article and get a disease report
<b>Acceptance Criteria</b>	<ol style="list-style-type: none"><li>1. User input the article ID and send a request to our API server</li><li>2. A JSON response will be return, which includes disease report for the particular article</li></ol>

## 4. System Design

### 4.1 Backend Design

#### 4.1.1 API Protocol

REST

#### 4.1.2 Independence

Any client will be able to call on our API using standard GET protocols as long as they have followed the proper input instructions. We will try to make the endpoints as self-descriptive as possible so the client will need no knowledge of the internal workings of the API.

#### 4.1.3 Versioning

We will be using URI versioning for all working iterations of our API by adding a unique version number to the URI of each resource. This will be done to not break programs which are reliant on older versions of our API. Documentation will be provided for each version through Swagger, and all functionalities should be self-descriptive.

#### 4.1.4 RESTful

Our API will follow the RESTful principles; most importantly, our API will be stateless and will have a uniform interface. All resources in our API will be identified by a noun which mirrors the name of our database collection. Specific resources can be specified by calling upon the document's unique ID. Specific parameters for requests will be defined further in the report. For security reasons, we will only be allowing GET requests to go through our API.

Some parameters are passed as a part of requested URL (e.g. ID of the articles), while for some are difficult to be encoded in the URL, they should be passed in the body of the request (e.g. key terms). More information has been provided in the sample request section.

Content-Type will be limited to "application/json" with the majority of the request parameters being in the body and headers of the request.

#### 4.1.5 Deployment

We will be deploying our API on the Google Cloud Platform as it allows us to easily develop, protect and monitor our API. We also have the option to include JSON Web Tokens and Google API keys in the future. It is also flexible with frameworks and allows us to adapt when we design our web platform.



### 4.1.6 Testing

In order to prevent any exceptions. We will have some autotest function to check if there is any error during data transmissions such as no response and missing data. Also, input values would be checked before the user request. (see 7.1)

## 4.2 Front-end Design

### 4.2.1 Modularization

Since the usage of React, developers were able to decompose the application into components. In this way, pages can be partially reloaded and hence provide more fluent user experience to the user.

### 4.2.2 Data-Driven

The state of some components in the application is determined by data which is stored under the namespace of the component. Once the data is changed by asynchronous operation, the rendering change will be triggered accordingly.

### 4.3.3 UI Library

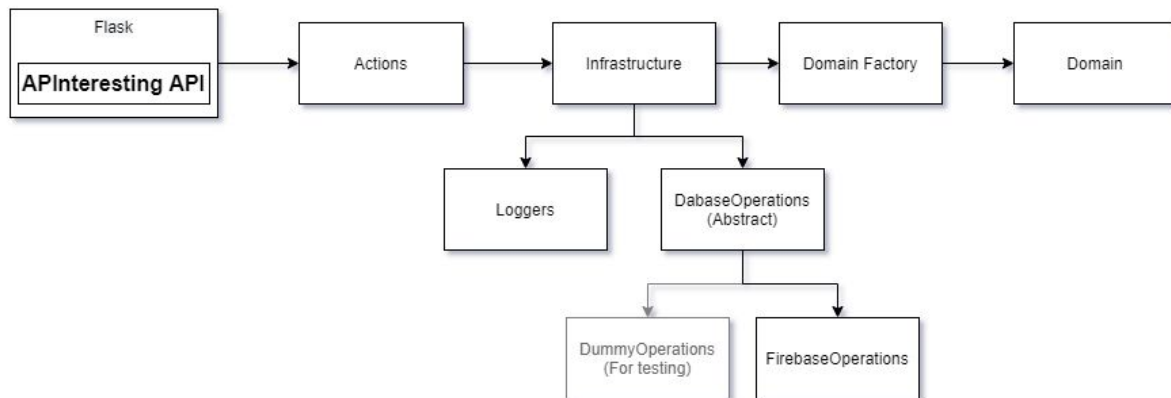
- Material UI
- Semantic UI
- W3 CSS
- React Bootstrap
- CanvasJSChart
- React Loader Spinner

We are using Material-UI as the major front-end library to design front-end components. And Semantic UI and W3 CSS as assistant libraries to help with designing some of the small components. React Bootstrap is used to locate the component in the right position. Material UI and CanvasJSChart are used to produce interactive charts. React Loader Spinner is used for its elegant loading animation.

## 4.3 Software Architecture

### 4.3.1 Backend Structure

#### 4.3.1.1 API Structure



The API code is nicely decomposed into multiple classes defined under multiple namespaces following the Single Responsibility Principle and the Open-Closed Principle.

Each class defined in Application\Actions acts as a facade representing a specific type of task, comprehending the operations that took place in the lower levels.

Classes in Domain namespace are atomic classes representing the smallest and the most atomic infrastructure of objects. In this way, we created an easy way to map the data in the object and the data in the database.

Classes in the DomainFactory act as adapters carrying the task of converting data from the database, which normally in a directory format, to the initialised object from the Domain namespace.

Classes in Infrastructure commonly involve IO operations. An example of the satisfaction of the Open-Closed Principle here is the DbOperations and DummyOperations namespace. Functionalities included for them are almost the same regarding the effects, except the DummyOperations are only generating meaningless simulating database operations used in early development phases when the actual database operator classes have not been completed. Hence, when the database operations are finished, it is very easy for any developer to change the code to use the real database, also easy to adapt to some other databases like MongoDB or MySQL.

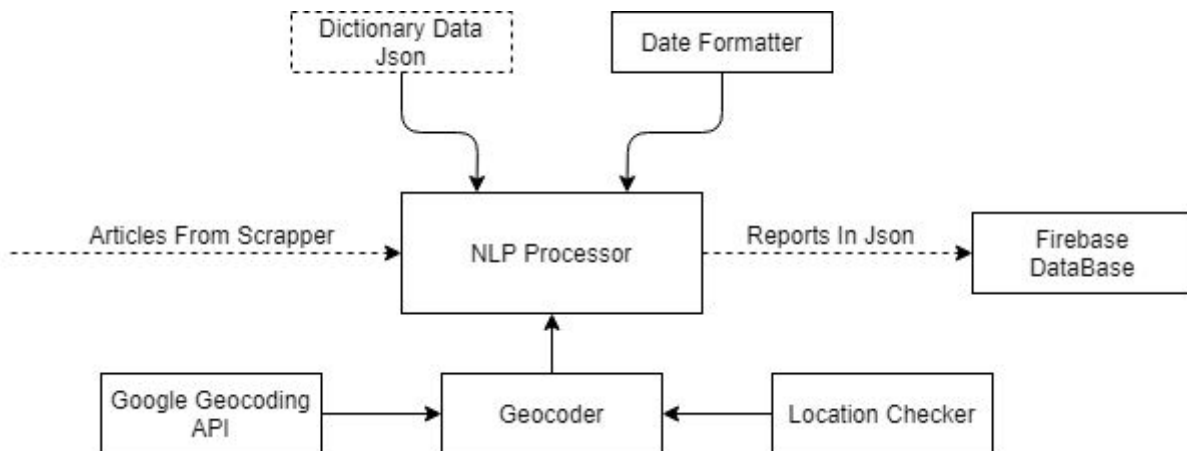
#### 4.3.1.2 Scrapper Structure



The graph above intuitively shows the structure of the process of our data source collection. The scrapper code is decomposed into multiple classes to handle each independent functionality followed by the principle of the Single Responsibility Principle and Open-Closed Principle.

Scrapper class is focusing on scraping the HTML page from the data source, and further passing into the filter class, which utilizes the BeautifulSoup Python library, To filter the important and useful data.

#### 4.3.1.2 Nature Language Processor (NLP) Structure



The graph above demonstrates the workflow and the relationship between sub-component. Dashed arrows represent a workflow. Solid arrows represent composition relations. Dashed rectangle means the file is plaintext in JSON format, otherwise, it is an executable file.

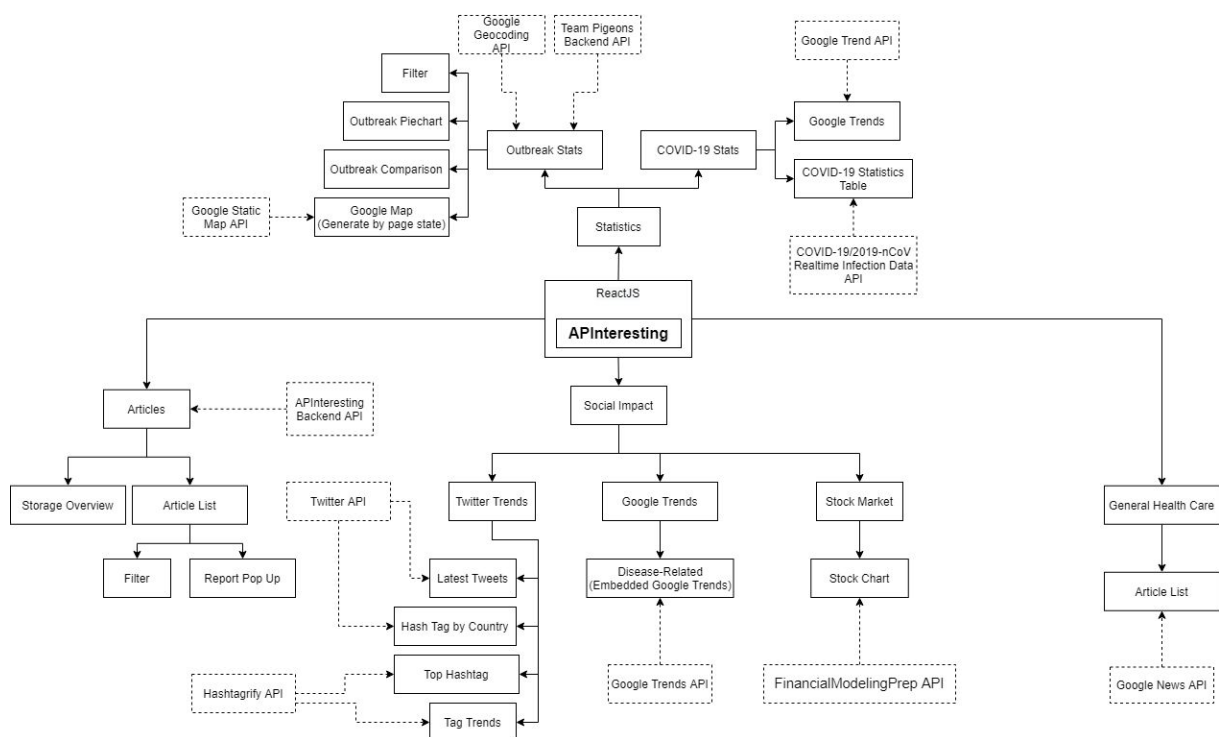
NLP Processor will take in the text of an article from Scrapper Component (See section 4.3.1.2) and return our report of that article in JSON format to the database. It will import dictionary data and use them to filter keywords. With date information and location information provided by the sub-component, it will then try to separate all information into multiple reports if that's the case and send the final result to the database.

Date Formatter uses regular expressions intensively to extract the date and time information of the text and convert them into the required formal format.

Geocoder will filter for location names from the article. It will then geocode them and try to give out the full address in a string for each of the entries. Google Geocoding API is used for geocoding and generates address strings. Location Checker will need to tell if a location name is a country or a city name or other. Geocoder will return a list of locations in the required format to NLP.

Dictionary data is in JSON format, including `disease_pattern.json`, `disease_category.json`, `search_pattern.json` and `syndrome_pattern.json`. Combined they provide the knowledge and keywords set for NLP and can be changed easily to make the NLP component pretty flexible.

### 4.3.2 Front-end Structure



The graph above intuitively displays the structure of our front-end. Note that the boxes with dashed border represent the API usage in the components.

The philosophy of designing the front-end in this project is to satisfy the Single Responsibility Principle. We attempted to decompose the whole application into the combination of smaller components. For example, the components with the suffix “Page” at the end will be focusing on the rendering of a skeleton page. A page component can have one or more smaller components carrying more rendering tasks like chart rendering or data fetching.

#### 4.3.2.1 Statistics Component Structure

Statistics component has two web pages, one is Outbreak Statistics, one is COVID-19 Statistics.

In the Outbreak Statistics Page, we fetch Team Pigeons API for outbreak data and use Google Geocoding to obtain the geocoding of each mentioned location. Users can specify search criteria and we will show the new outbreaks over time, accumulated outbreaks over time and use a map to show the locations of each outbreak. Google Static Map API is used for generating a Map with labels. There ought to be a comparison function to enable users to compare the statistical differences between country and global world.

In COVID-19 Statistics Page, we fetch the latest data from COVID-19/2019-nCoV Realtime Infection Data API about COVID-19 confirmed/suspected/cured/death cases of each country and display them to users, meanwhile, enabling them to export data. In addition, we use Google Trend API to provide the keyword search interest over time and trending questions around COVID-19.

API Usage: Google Static Map API, Google Geocoding API, Team Pigeons API, Google Trend API, COVID-19/2019-nCoV Realtime Infection Data API

#### 4.3.2.2 Social Impact Component Structure

Social impact component has three pages, Twitter trends page, Google trends page and Stock market page.

In Twitter trends page, Data sources for Latest tweets and Hashtag by country are provided by Twitter API. Hashtag by country will filter the TOP 50 popular hashtag in one country to get disease-related tags. Latest tweets will fetch the latest disease-related tweets. Top Hashtag (global) and Tag trends are supported by Hashtagify API. Top Hashtag contains a table of 10 disease-related hashtags with details while Tag trends contain the historical trend of several disease-related hashtags.

In Google Trends Page, we embed a list of disease keywords and their corresponding search popularity value. The data is supplied by Google trends API.

In Stock market page, we use data from FinancialModelingPrep API to get a series of real-time Industrial Averages and display them in the graph with total confirmed cases in global to demonstrate the relationship between pandemic and economy.

API Usage: Google Trends API, Twitter API, HashTagify API, FinancialModelingPrep API

#### 4.3.2.2 Articles Component Structure

The articles come from our API - API-Interesting, we also use our API to generate disease reports for each article. In the disease report, we provide the function, which is exporting the report as JSON. We also use Google geocode API to locate disease location and show on Google Map. In addition, we provide a search bar to allow users to search for articles by filters. The filter includes start date, end date, article location and disease keywords.

One important point we would like to point out, the articles list are shown by the order of frequency of the disease keywords. This design is beneficial for users to explore the most important article.

Furthermore, we have used a pie chart to show the articles storage for the top 6 diseases in our database.

API Usage: API interesting API

#### 4.3.2.2 General Health Care Component Structure

General health care component has a relatively straightforward construction. It contains a list of live health care tips fetched from Google News API. In addition to that, It contains some general tips in protecting the public from the pandemic.

API Usage: Google News API

## 5. Technology Stack

### 5.1 API implementation Language

#### 5.1.1 Final choice

Python 3 + Flask

#### 5.1.2 Justification

Java and Python 3 are considered. Because of the abundance of libraries and simple code structure, Python 3 has been chosen as the implementation language. The usage of Python 3 makes products can be delivered in shorter development time. Another consideration for Python 3 is although all members have used Java before, we are more confident with Python 3 and Flask as they have been used more frequently in our previous studies.

#### 5.1.3 Encountered Challenges

- Change of requirements always happened throughout the development process. Every time a change is required, part of the code needs to be reviewed and rewritten, although as the API has been partialised into many classes, most of the changes did not involve excessive file changes.

#### 5.1.4 Limitations

- The concurrency performance of a Flask application is not very outstanding. Therefore, it is expected that when there are more clients trying to fetch large amounts of data from the API simultaneously, the queuing time could be larger and hence the performance is worse. This may be resolved by running the API on a better server using server applications that support multiple threads/works like Gunicorn.

### 5.2 Database Component

#### 5.2.1 Final choice

Firestore

#### 5.2.2 Justification

We first considered MySQL/PostgreSQL, MongoDB and Firebase then chose Firebase.

For MySQL and PostgreSQL, as they are both relational databases, much time will need to be spent on designing the schema according to our previous experiences on databases.

Also, what has been scraped will also need to be stored as a more atomic form to prevent performance loss during querying. However, JSON can be directly read and stored into the document-based database without reducing too much performance.

Compared to MongoDB, Firebase is more light-weighted, so one can use Firebase without too much deployment on the server. We also decided that Firebase would be more suitable as Phase 2 of the project required a platform. Firebase is easily integratable with Google Cloud Platform and will give us some experience with the Google Cloud Platform functions when the time comes to design our platform. Considering the scale of this project, we finally decided to use Firebase.

### 5.2.3 Encountered Challenges

- Pipeline filters. Firebase is a lightweight database and can only support one `array_contains()` function per query, this makes pipeline filters hard to apply on and make a recursive search for keywords impossible. (e.g. articles go through location filter AND date filter AND multiple keyword filters) This is solved by altering the data structure and limited keyword filter to one. We create a list of locations at each article JSON to search on. Multiple keyword filtering process will be handled at API.
- Date Comparison & filtering. Dates are hard to compare as strings. We changed the date from string to integer timestamp to filter time.

### 5.2.4 Limitation

FireBase has a daily quota limitation capped at 5.5k read and write, we need to minimise the communications between API and database and do more filtering process at the database stage. In addition, if the number of our users increase exponentially our application might hit this threshold.

## 5.3 Text Mining Component

### 5.3.1 Final Choice

spaCy (Python library)

### 5.3.2 Justification

SpaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. It provides extremely fast and accurate trained models for Named Entity Recognizer Feature which we used to pick out keywords of different categories (date, time, countries and cities). It also provides the Phrase Matcher Feature for us to match potential disease and syndrome patterns from text in  $O(n)$  time-efficiency. Furthermore, it holds potential ability in providing us with a customized Natural language processor trained by our team.



Our previous plan is using third-party API like Google Natural Language API and Rapid Text Analysis API to analyse the articles and obtain data. However, from both commercial and performance considerations, we choose to use python library at last. Google API will charge us for massive API requests, and the average response time is around 0.3 second per request which is huge compared to library functions.

Among all NLP libraries (NLTK, spaCy, TextBlob), we choose spaCy as it provides more powerful features and faster algorithm in processing text. Unlike NLTK and TextBlob, which is widely used for teaching and research, spaCy focuses on providing software for production usage.

### 5.3.3 Encountered Challenges

- Selection between vast amounts of NLP python libraries. There are a vast number of NLP libraries with distinctive focus and different ease levels to use. We read through their documentation and sample usage and finally chose spaCy as our NLP library. This is a one-way road as it is hard to roll back for any reason (e.g. provides too fewer functions or hard to manipulate)
- General names for diseases and syndromes. The name of disease or syndrome appearing in the text can be very different from its formal name. We match all other names of a disease or a syndrome to their formal name by a lookup table. The lookup table can be loaded from outside as a JSON file thus easy to change or update.
- Massive format of date, time strings. The date format can be very different between reports (e.g. dd/mm/yy, yy/mm/dd, yy-mm-dd) and sometimes can be a range of dates (e.g. Feb 3-5). This is solved by a two-step process. First, spaCy will extract date-like patterns on a linguistic approach, then vast regular expression catchers are used to capture date information and then recombined into the desired format.
- Unexpected captured patterns from spaCy. spaCy holds accuracy at 86.08 on Named Entity Recognition features, relatively high compared to other NLP libraries but we need to capture all these errors. We create our own program to validate all country names and again vast regular language catchers are used to capture unexpected patterns from spaCy.
- Separate Multiple Reports from an Article. We can successfully capture all diseases and syndromes in an article but if multiple cases are mentioned in an article, It is hard to separate them into different reports. This is partially solved by a lookup table and analysis on language features to distinguish very different diseases and most matched syndromes that are mentioned in an article.

### 5.3.4 Limitation

- Limited to English currently, only English text can pass the filter and get processed into JSON. In addition, spaCy is a young python library and can only support seven languages.

- The lookup tables need to be imported from outside and they are created by humans. Therefore, the NLP processor cannot do deep learning and improve itself automatically
- Separation of multiple reports from one article can only be applied to diseases and syndromes. We cannot allocate date and locations to different reports

## 5.4 Scraper Component

### 5.4.1 Final Choice

Scrappy, BeautifulSoup, HTTP requests

### 5.4.2 Justification

Scrappy is used to scrape the HTML file from a website, then we use BeautifulSoup as a filter to extract the useful data from the HTML file. Scrapy has been chosen as it is simple to use, fast and extensible framework for web crawling, web scraping. In addition, Scrapy is easier to build and scale large crawling projects, and it automatically adjusts crawling speed using Auto-throttling mechanism. We choose BeautifulSoup as our HTML filter since it uses the lxml format to extract HTML data, which is very fast and memory efficient. In addition, it is also simple to use and extensible.

In addition, we use HTTP requests to scrape some easy-access web pages. This improves the scraping speed and simplifies the implementation. and the advantage of HTTP requests is it returns a viewable response, so we can use the response to filter some invalid URLs.

### 5.4.3 Encountered challenges

- Invalid URLs to scrape. After scraping a large number of web pages from flu-trackers. We see that flu-trackers sometimes return some websites that are unable to be accessed by scrapers. so that we have to do further filtering to filter out the invalid HTML
- Filtering data from HTML by BeautifulSoup. Since the web pages have variable CSS class names, it is very hard to filter data by consistent filter configuration. But this is no way to solve since all the web pages have different and various CSS names. So we have to do a trade-off between only extracting the accurate data and extracting some useless data.  
In addition, Since we use consistent filter configuration to extract data, therefore if some HTML files do not have these attributes, we would lose some key data. To solve this we have to put further exceptions for missed data. This is a challenge since this situation is rare, so we discover this problem after scraping many web pages.
- Scraper can only scrape one page at each script call. The scraper in our implementation is not able to scrape multiple pages in one script call. therefore we have to store URLs into a .txt file and use an extract script to run scraper for multiples pages scrapping

#### 5.4.4 Limitation

- In flu-trackers, there are many source URLs for each post that are not able to access. So currently we only have around 1000 data in the database. This is also a problem when we update new data into the database. From current experience in updating the data from flu-trackers, about 40% of new source content URLs are not able to access or invalid, so this would limit the speed of extending our database size

### 5.5 Geocode Component

#### 5.5.1 Final Choice

Google Geocoding API

#### 5.5.2 Justification

Google Geocoding API is relatively easy to use and is vastly used in different software applications. Google Geocoding API can transfer the name of a place into a unique id. The response also includes formatted address names and the names of administrative areas where that address belongs to. This enables us to support geo-location taxonomy. (e.g. search for NSW articles and can return Sydney articles). In addition, Google Geocoding API has strong error recovery function and autocomplete function.

The other potential pick is geolocation python library. It is easier to use and the responding time will be way faster than requesting API. However, geolocation python libraries only support a limited number of countries (doesn't support China) as locations and the functions provided are awkward to use. In the other side, Google Geocoding API also has a fairly complete location database thus being able to recognize locations that are not very common, most developers have experience in using Google Geocoding API, therefore Google geocode is more popular and it will be easier to implement the "location to geocode" function in our system.

#### 5.5.3 Encountered Challenges

- Duplicate location names. We prefer to have more accurate location addresses to be stored in the database. For example, if "Wuhan, China" has already been detected, China should no longer be captured as Wuhan is more accurate than China. Meanwhile, calls to Google API are expensive both in money and time. This is compromised by a two-step process, all location-like patterns will be checked to see if it is a country name or others and then separated into two lists. The location list will go through the Google Geocoding API first and their country names got stored in another list. Counties which duplicates in that list will not be processed.

#### 5.5.4 Limitation

- Relatively expensive charge. Google charges \$5 per 1000 requests, in average each article will cost \$0.2 as processing fee.
- Performance, request for Google API takes time and is comparatively large to other components. Average time for an article to be processed is 0.2 sec when geocoding is off, and It is around 2 seconds when geocoding is used.

### 5.6 Operating System

Linux

Due to the limitation of Gunicorn, we can only deploy our application on Linux.

### 5.7 Server Application

#### 5.7.1 Final Choice

Gunicorn + Caddy

#### 5.7.2 Justification

Gunicorn has been chosen because of the easy configuration and good performance on Flask websites. We considered only using Flask as the server application, but it could be too unstable to be used on the actual product. We added Caddy into the stack as the process of the project to satisfy the requirement of hosting Swagger documentation along with the API. With the usage of Caddy, the API endpoint and the documentation can be deployed separately as two different websites so that reduces the need to integrate swagger into our API.

#### 5.7.3 Limitation

- Gunicorn can only be deployed on Linux, and hence the flexibility of our application is restricted to some extent.

### 5.8 Front-end Framework

#### 5.8.1 Final Choice

ReactJS

### 5.8.2 Justification

React has been chosen over other libraries because of its ease of use. Its modular system allows us to maintain and update code easily. React allows us to reuse the code which can really boost developer productivity. In addition, React is designed with performance, such as server-side rendering and virtual DOM, allowing us to create large-scale apps which are really fast. Furthermore, React is a reliable and up-to-date technology, since it is supported and maintained by many big companies such as Facebook, as well as many independent contributors worldwide. Moreover, most of the members in our group have already used React before or are familiar with Javascript so learning React came naturally. One other option that we considered was VueJS, another popular Javascript Framework. However, we decided to go with React because of our previous experience with it in SENG2021 as well as React having more mature libraries and a larger ecosystem.

### 5.8.3 Limitation

- The learning curve for React is very steep. Some of the members did spend some time learning the usage of React and the format of JSX in order to be able to start developing.

## 5.9 Front-end Library

### 5.9.1 Final Choice

Material-UI

### 5.9.2 Justification

Material-UI is a mature and well-designed front-end library, which has been popularly used in products developed by Google. Bootstrap is considered but compared to Material-UI, it provides less preset components even though they provide almost the same basic structure. Also from the first glance, Material-UI looks more energetic.

### 5.9.3 Limitation

- The usage of libraries somehow increased the difficulty of partially customising the page as the interfaces between some predefined components may interfere with each other.
- The documentation for Material-UI is complicated to follow. Members spent plenty of time trying to understand the structure and the usage of some components.

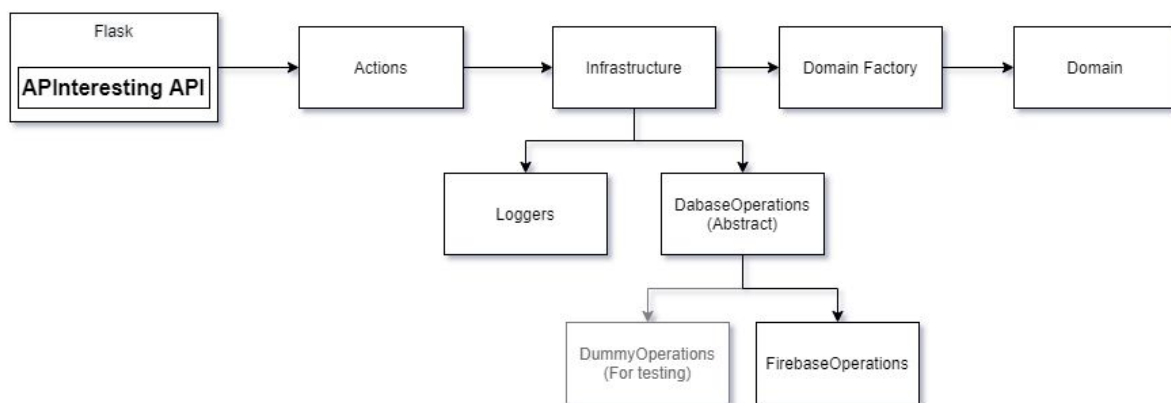
## 6. System Implementation

### 6.1 Backend Implementation

#### 6.1.1 Client Identification

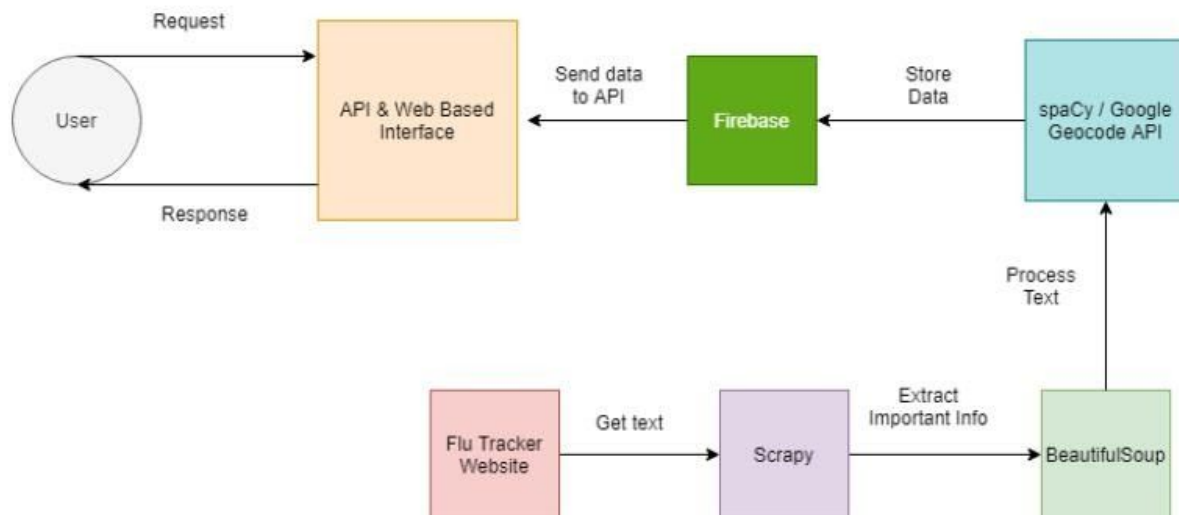
A client should attach “identity” as a custom HTTP header in every request to the API, otherwise, the API will return a response with 401 status code to notify the client to do so. This value should typically be the group name. It will not be validated in the current version but it will be logged. This method of authentication is trivial under the scale of this project. However, it can be useful to extend it to Token, which is similar to the JSON Web Token Authentication method, in the real production environment, especially when there are multiple clients using this API.

#### 6.1.2 Basic Flow



When a request arrives, a hook, where the authentication check will be done, provided by Flask will be triggered to check the validity of the request. After passing the check, the router will call the corresponding facade method with appropriate parameters. The facade method will perform another check on the arguments. If the check is passed, the facade will initialise or call a database-related class to retrieve information from the database. The result will be converted to initialise an object in the database object by a domain factory and returned all the way up to the response. Before responding to the request the response will be transformed again for a final decoration to attach some meta-data such as data source, request time and our group name, as well as convert to JSON.

### 6.1.3 Data Processing Flow



All information will be given unique IDs and stored on Firebase in the form of a document. When a user queries our API, the appropriate document will be selected from our database and sent back to the user in JSON format.

Firstly, HTML from the Flu Tracker website will be scraped by the Scrapy library. The extracted HTML text will then be processed through BeautifulSoup to get the text from the HTML elements that are pre-specified in the script. Each section of text will be parsed using the Google NLP API to extract key information such as locations, diseases and syndromes to create the reports. In the end, the locations will be further analysed by the Google Geocode API to obtain the google geocode ID.

When those information above is stored into the database, they will be given unique IDs by Firebase before stored in the form of a document. When a user queries our API, the appropriate document will be selected from our database and sent back to the user in JSON format.

### 6.1.4 Data Storage

All the data for our API is stored with Firestore. Firestore is a no-SQL database provided by Firebase, a branch of Google. Each document in our database collection has a unique ID and relevant metadata such as `created_At` and `updated_At`.

For each document, we store:

- URL of report
- Extracted disease keywords
- Extracted syndrome keywords
- Frequency ranking of keywords
- List of generated reports
  - Extracted locations from the report (along with geocode)

Using Firestore gave us a few challenges. These included the filtering of data and daily quotas. Firestore has a limit on how many queries we can chain into a request. This made querying nested data structures hard and forced to use some unorthodox ways of storing data.

For example, we stored our keywords as:

```
"keyword_list": {  
  "sars": true,  
  "acute respiratory syndrome": true,  
  "covid-19": true,  
  "virus": true,  
  "outbreak": true,  
  "epidemic": true,  
  "emerging": true,  
  "infectious": true  
}
```

Where in our query we would need to compare our keywords to true. Furthermore, the free version of Firestore has a daily request quota of 50k requests. This forced us to deprecate our `getAllReports` function as repeatedly reading the thousands of reports in our database would make us exceed our quota.

### 6.1.5 Logging

This API will generate one line of log in the backend file system. With the format of `[Time] | [Identity] [URI] [Response Status Code] [Execution Duration]`.

The logger is triggered by the hook provided by Flask. The server will mark down the time when a request arrived. By the end of the execution, the execution duration will be calculated. The server will extract other data from both the request and the response to generate and write down the log in the file system.



### 6.1.6 API Usage

- Firebase
- Google Geocode API

### 6.1.7 Library Usage

- SpaCy
- Scrappy
- BeautifulSoup

## 6.2 Front-end Implementation

### 6.2.1 User Interface

Most of the elements used on the page have been predefined in Material-UI library. The grid system provided by the library has been used to better separate a page into sections. Also, by using these preset components, the time spent on designing pages with a consistent visual effect is significantly reduced. CanvasJSChart and Material-UI are used to provide interactive graphs and charts to display information. CanvasJSChart is also responsible for providing the export feature for graphs that are created. React Loader Spinner is used to create loading animation in order to make the interface more user-friendly.

### 6.2.2 Data Visualisation

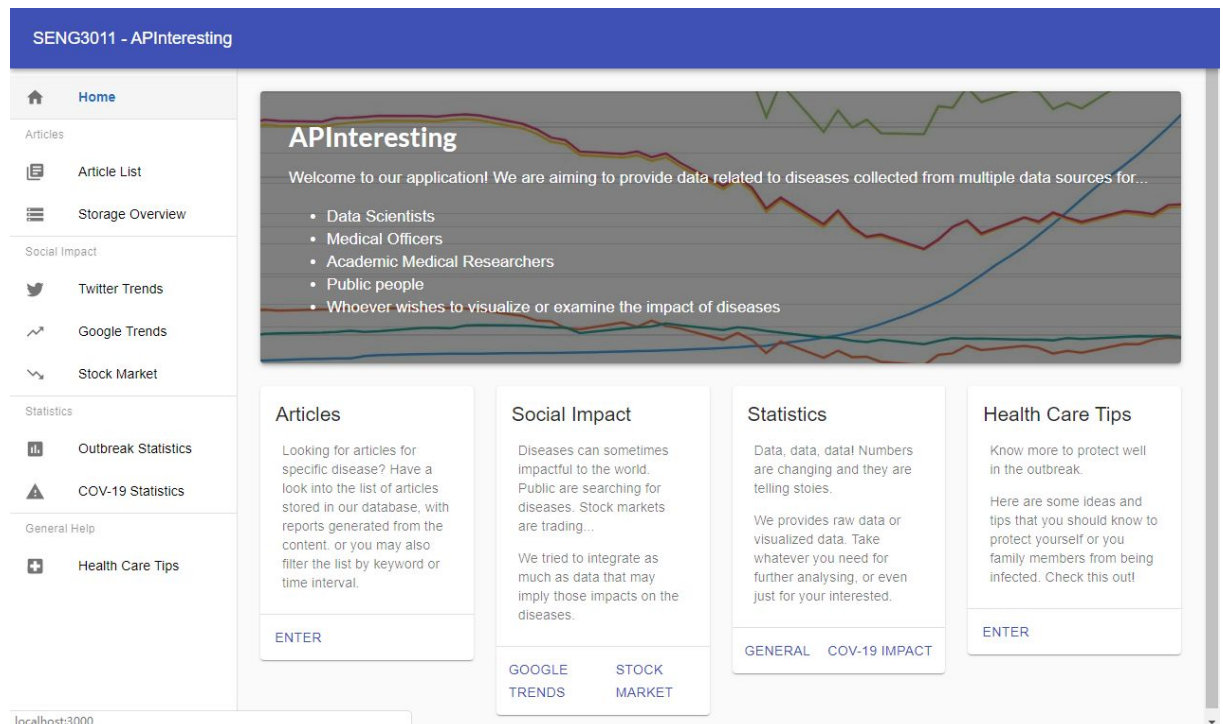
Most of the charts are rendered using ChartJS. In order to make the chart respond to user behaviours such as filtering and searching, most charts are implemented in a data-driven way, which means the chart is bound with the state of a React component, once there is a change is detected in any state property, the component should check the necessity of updating some other property, as well as the rendering.

### 6.2.3 API Usage

- Google News API
- Google Trends API
- Google Geocoding API
- Google Static Map API
- Twitter API
- Team Pigeons API
- FinancialModelingPrep API
- COVID-19/2019-nCoV Realtime Infection Data API
- Hashtagify API

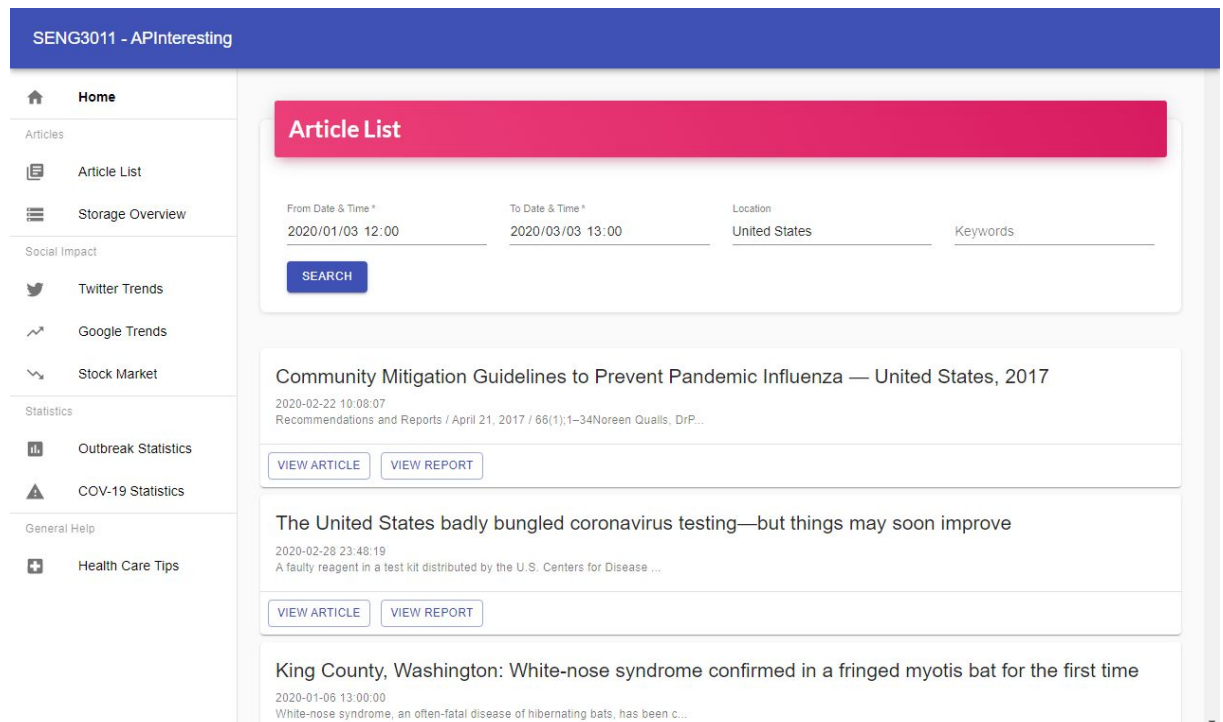
## 6.2.3 Interface Walkthrough (Screenshots)

- HomePage:



The Homepage gives a brief description for each page and provides user entry to navigate to the page.

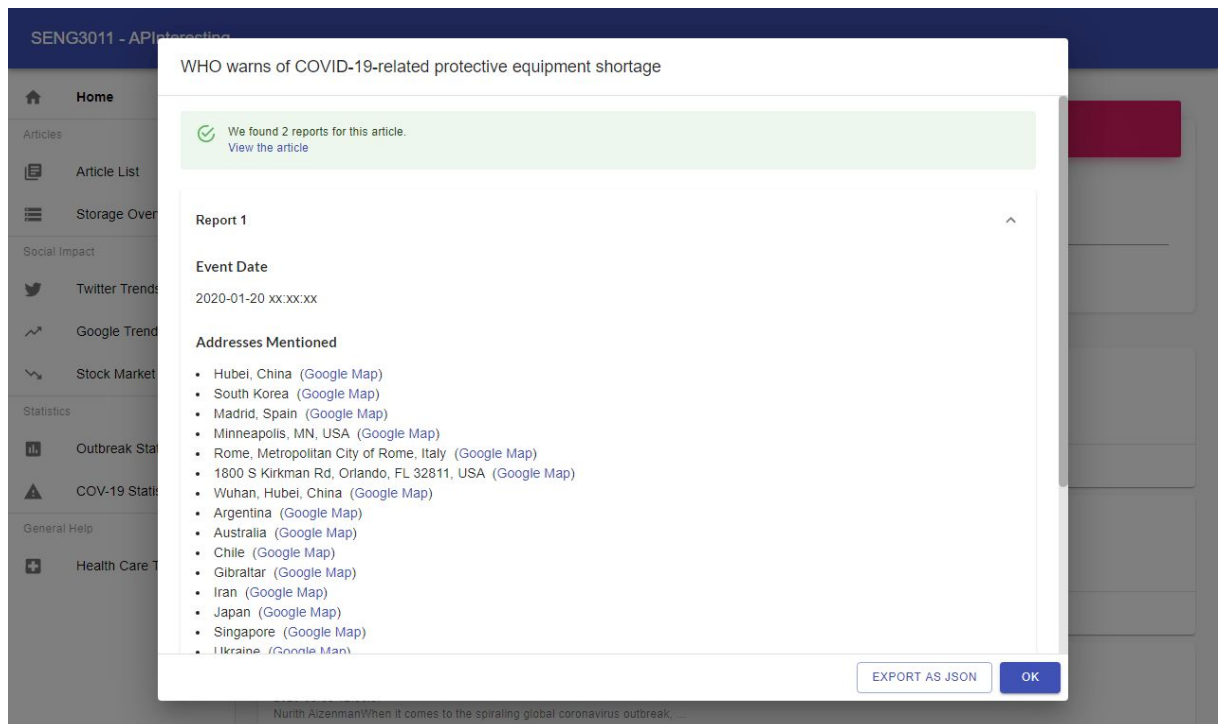
- Article List Page: (Requirement 1, 4)



Article List Page enables the user to search articles on a specific start date, end date, location and keywords.

After clicking the search button, the user can observe a list of articles.

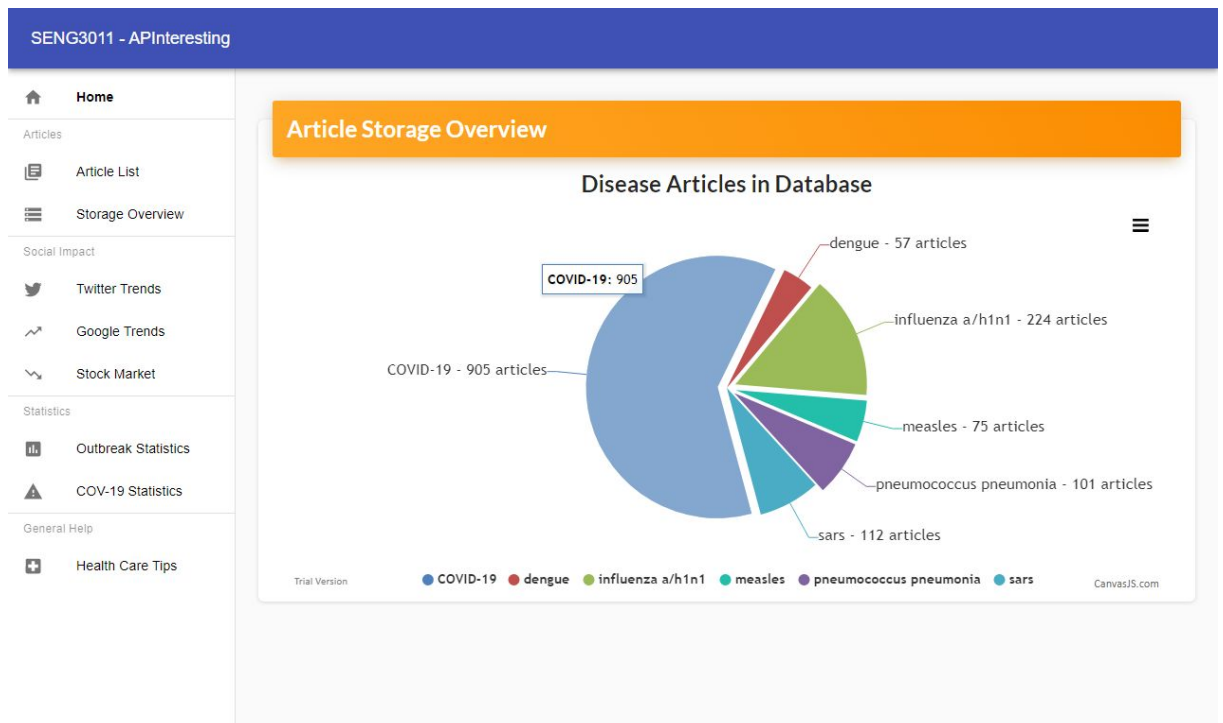
User can view the origin website or view reports that are extracted from the article by clicking buttons.



After click “View Report” button, the user can see how many reports are extracted and the details of the report.

User can also click the highlighted address to go to Google map for location details. Click “Export” will download the JSON format of the report.

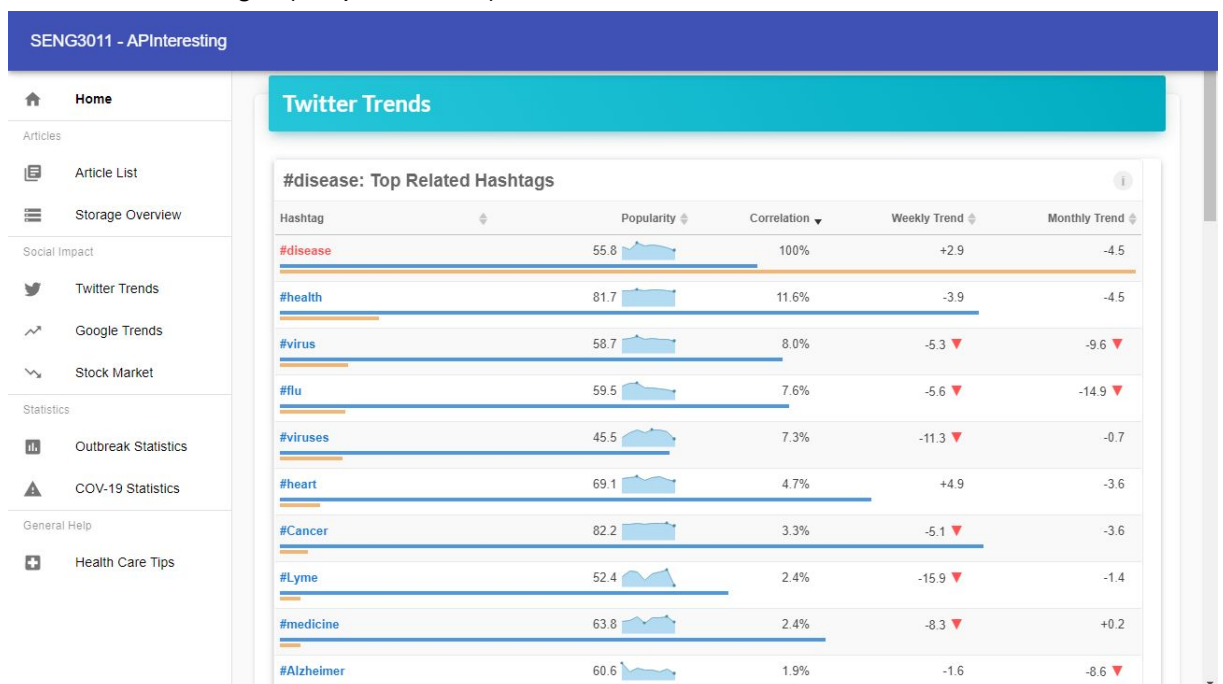
- Storage Overview Page: (Requirement 1)



The Storage Overview Page contains information about the number of articles in the database.

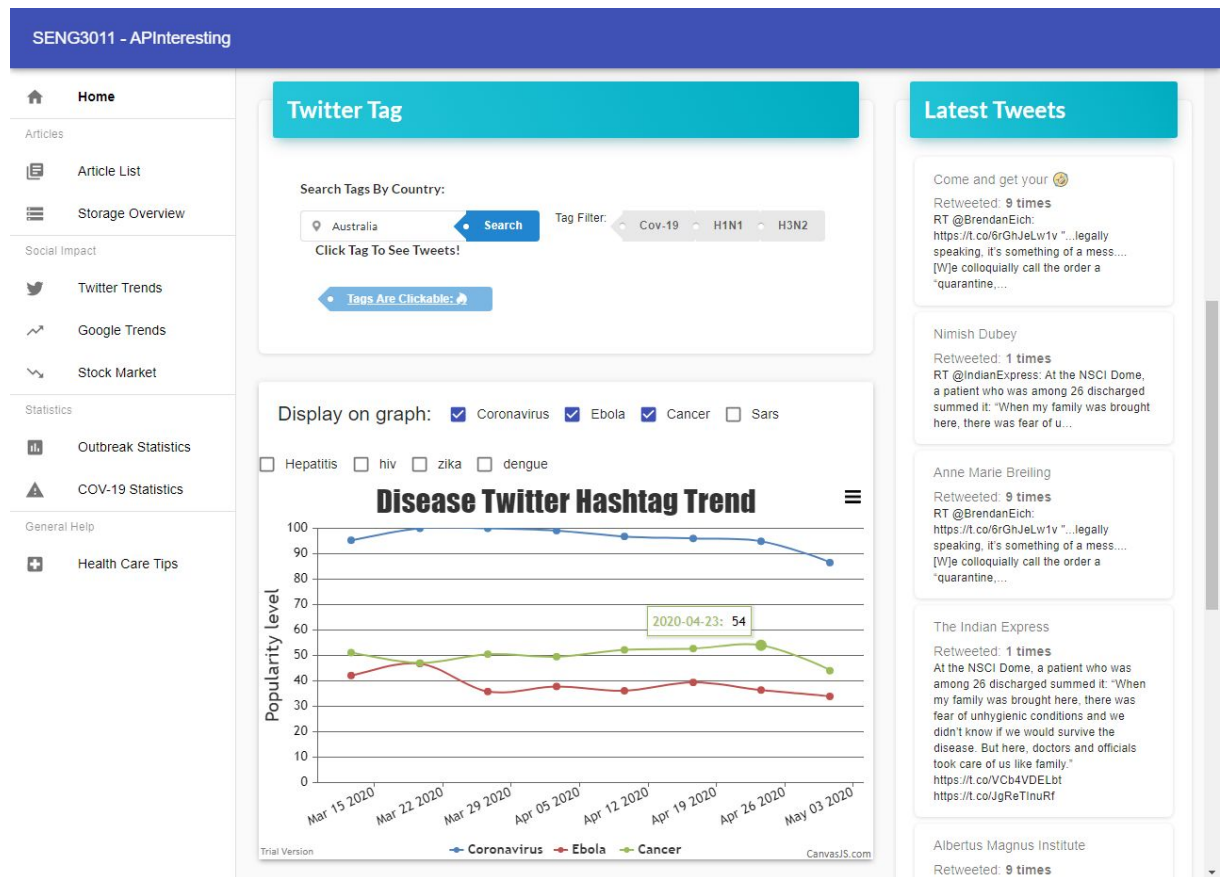
User can see the exact number of articles of different category by hovering the mouse. The graph can be exported by clicking the button.

- Twitter Trends Page: (Requirement 6)



User can see a list of hashtags and their popularity, historical popularity graph, correlation degree to disease and their monthly/weekly trend situation. This can show if a hashtag is being more popular or less popular.

User can also click the tag to sort or resort all entries.

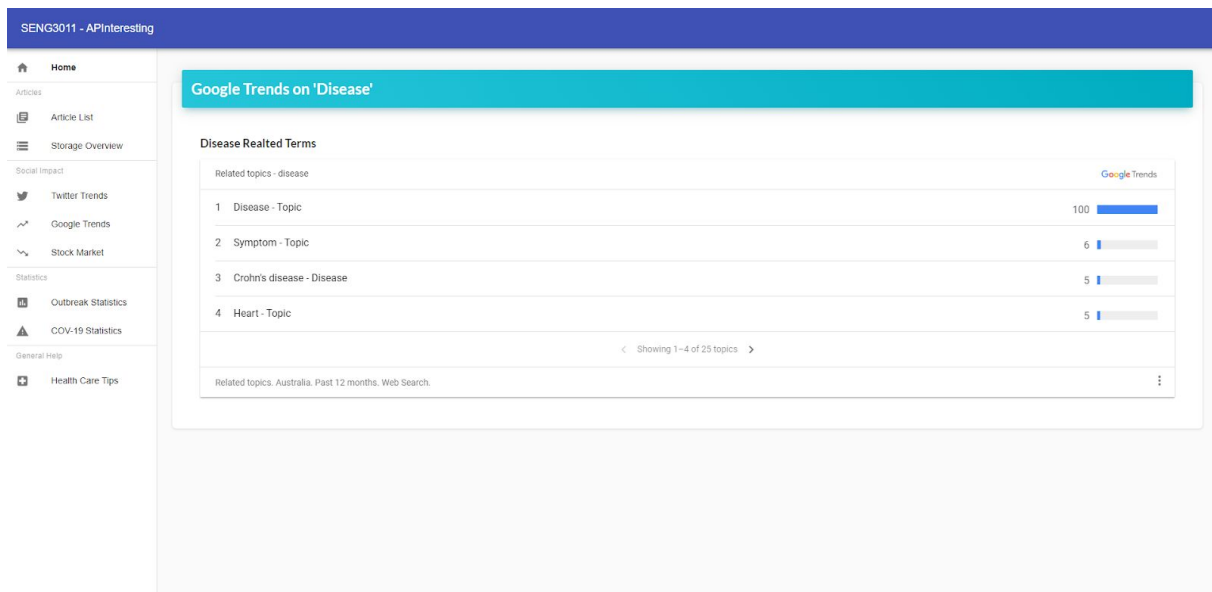


User can search countries for popular disease hashtags and use tag filters to restrict search results. The tag is clickable and can lead user to specific twitter page.

User can also find trend history of disease tags on the graph and use the checkbox to choose which diseases to display. The graph can be exported by clicking the button at the right corner.

User can see latest Tweets that are related to disease from twitter at the right sidebar.

- Google Trends Page: (Requirement 5)



Users can see the most searched terms related to the “Disease” keyword on google in Australia over the past 12 months

Users can click on the left and right arrows and view up to the top 25 most related keywords.

Users can click on individual rows in the table to be brought into a more detailed trends view supplied by Google.

- Stock Market Page: (Requirement 8)



User can see the Industrial Averages compared with the total number of confirmed cases.

User can zoom in or out the chart when scrolling on the chart.

- **Outbreak Statistics Page: (Requirement 2, 3)**  
*Screenshots were not able to be captured due to the failure of Team Pigeons API at the time of writing this report after the final demonstration.*
- **COVID-19 Statistics Page: (Requirement 5)**

SENG3011 - APInteresting

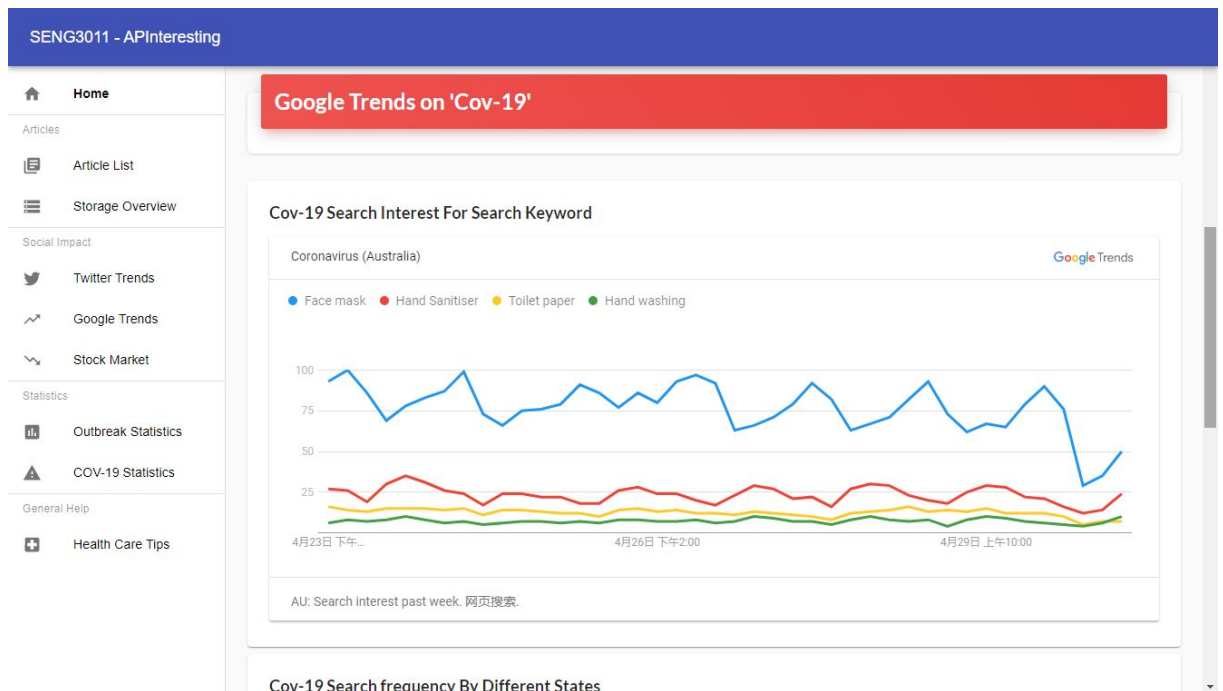
**COVID-19 Statistics**

Country	Confirmed	Suspected	Cured	Death
United States of America	1039909	0	120720	60967
Spain	212917	0	108947	24275
Italy	203591	0	71252	27682
United Kingdom	165221	0	344	26097
Germany	159119	0	123500	6288
France	128442	0	48228	24087
Turkey	117589	0	44022	3081
Russia	99399	0	10286	972
Iran	93657	0	73791	5957

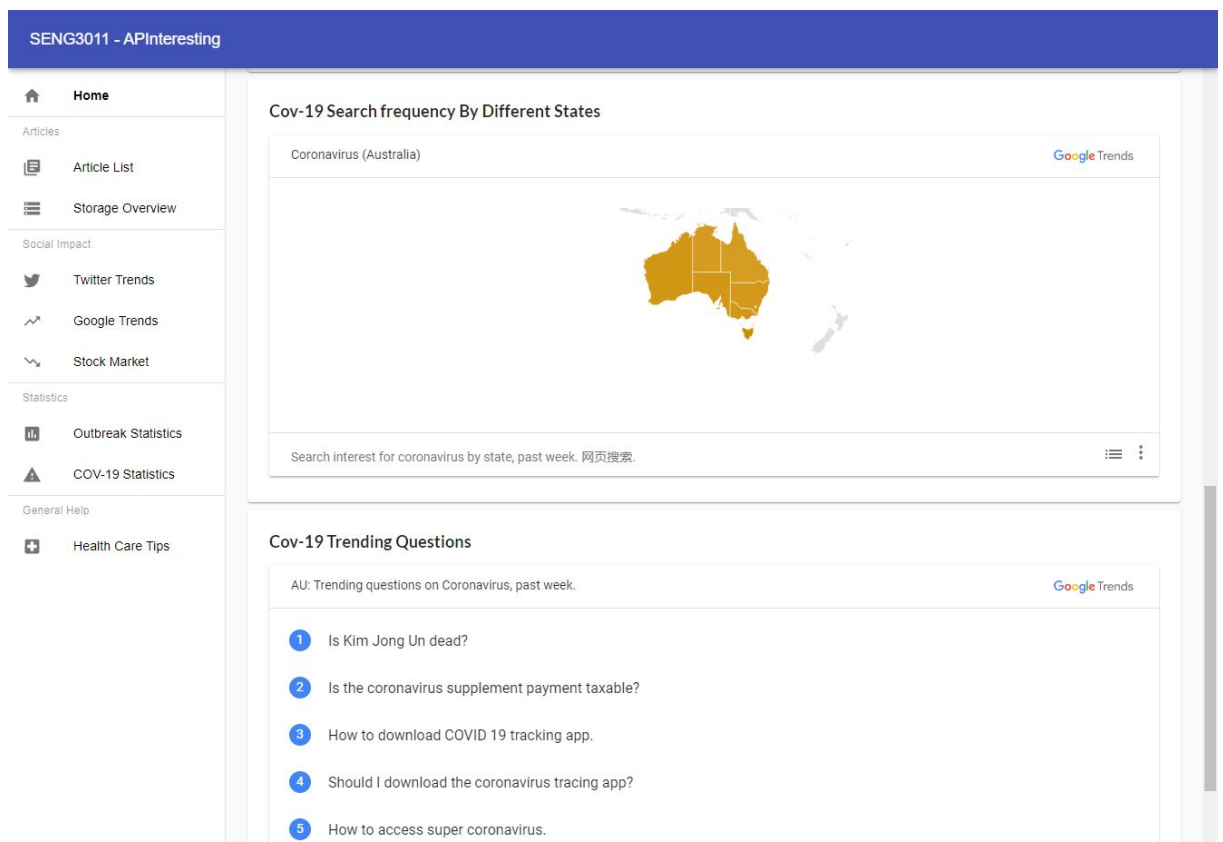
EXPORT AS JSON EXPORT AS CSV

User can observe a list of country with confirmed/suspected/cured/death cases and they are in sorted order.

User can click “Export As JSON” or “Export As CSV” to export the data for future study.



User can check the search interest of some keywords related to COVID-19 over time on the chart



User can check the search frequency of “coronavirus” overstates in Australia, and click the button in the right bottom corner to view them in list form.

User can also see a list a of trending questions related to COVID-19, and click them to go the specific search page



- Health Care Tips Page: (Requirement 7)

The screenshot shows a web application titled "SENG3011 - APInteresting". On the left is a sidebar menu with categories: Home, Articles, Social Impact, Statistics, and General Help. Under Articles, there are links for Article List, Storage Overview, Twitter Trends, Google Trends, and Stock Market. Under Statistics, there are links for Outbreak Statistics and COV-19 Statistics. Under General Help, there is a link for Health Care Tips. The main content area has a green header "Health Care Tips" and three article cards: "How to Access Reproductive Healthcare During a Pandemic", "Google will display virtual healthcare options in Search and Maps", and "Microsoft makes its advanced account protection free for healthcare workers". On the right, there is a section titled "Advices to protect yourself" with a circular icon of hands being washed and a list of five tips: 1. Wash your hands frequently, 2. Maintain social distancing, 3. Avoid touching eyes, nose and mouth, 4. If you have fever, cough and difficulty breathing, seek medical care early, and 5. Stay informed and follow advice given by your healthcare provider.

User can see Health Care Tips and Advice on this page.

User can also click the entry to go to the requested source news page.

## 7. Testing Summary

### 7.1 Testing Process Summary

Our application contains three main components that require testing, they are API & Database component, Scraper component and Data Process component. We use design by contract during development. The interfaces between components are invariant. Scraper component is promised to scrape valid date, headline and main text from HTML and passes into Data Process component, Data Process component is promised to return articles in a correct JSON format to API & Database component. Therefore, testing of different components can be done in parallel. Each component will focus on their own aspects and choose their own particular test data.

We use both manual and automatic testing mechanisms during our development. At an early stage, we use manual testing for debugging and improvement. After the component is behaving correctly, we build automated testing scripts by pytest to pick up unexpected errors and make sure every updated version is operating correctly. Manual test tools are in the local file folder and automated testing scripts are in Phase\_1/TestScripts folder as per the project specification.

### 7.2.1 API & Database Component

#### 7.2.1.1 Environment

- Debian 9
- Python 3.7
- requests
- pytest

#### 7.2.1.2 Process Summary

The typical process used in the testing is to make requests using the requests library, supplying specific and symbolic arguments to the API. Then use assertions supported by the pytest to check whether the API returns expected results.

When comparing the results with the expected result, what actually returns are not necessarily predictable sometimes. Considering this, some alternative handling methods have been used, which has been listed in the “Limitation” section.

#### 7.2.1.3 Overview of Test Cases

Firstly, test cases can be divided into two cases based on the validity of the input. For invalid input, the API needs to detect these invalid values and return error properly as per the Swagger documentation. For valid input, the API needs to parse and process the request correctly according to both project specification and Swagger documentation. Valid test

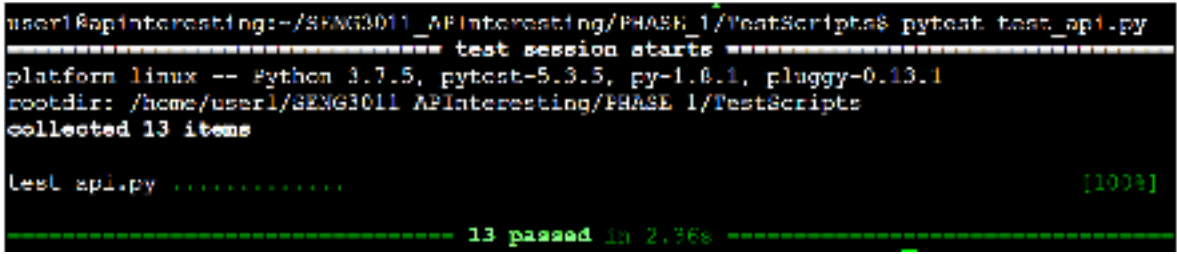
cases have been created for different endpoints, or even comparing each other between different endpoints to check if some data are actually identical if they are designed to be the same.

For example, when a client made a request without providing its identity in the header, the API should return an error with status code 401, along with the message describing the error. To test this, the test script simulates a request without identity in the header, then uses `pytest` to assert whether the response has a 401 status code and whether the response message is appropriate.

Similarly, if the client made a request with a valid date range filter, the API should return all articles within the date range provided, along with a 200 status code. To test this, the test script also simulates the valid request using valid data, then every response article will also be checked whether the publication date is actually satisfying the filter. And the 200 status code will be checked as well.

Furthermore, there are some extra basic tests to check the correctness of functions in the database. Such as writing data into the database, extracting the data by providing keywords, or id. The result would not be checked in detail, it would return a boolean value of true or false or the exact not-null data.

#### 7.2.1.4 Details of Test Cases

Test case ( <b>test_api.py</b> )	Explanation
test_response_format()	To check if the API returns the appropriate format
test_not_exists_identity() test_not_exists_url() test_get_news_error_date_format1() test_get_news_error_date_format2() test_get_news_error_date_format3() test_get_news_all_error_date_format1() test_get_news_all_error_date_format2() test_get_news_all_error_date_format3()	To check the ability of API to resist bad/invalid user input so that to ensure the strength of the API.
test_get_news_correct() test_get_news_all_correct() test_get_all_location_correct() test_get_report_and_article_by_id() test_invalid_source_content()	To check if the API can handle the normal scenarios with valid user input to ensure the basic functionalities, which could be representative for most of the scenarios.
Pytest Result: 	

#### 7.2.1.5 Coverage

For all of the four endpoints, there are some test cases created for each of the endpoints. Some test cases may be cross-endpoints to increase the reliability of the testing. Most of the error scenarios should have already been covered by most of the test cases. Approximately, it can be estimated **over half of** the functionalities have been tested.

For the database, most of the functions have been tested except for query. Some exception cases were also covered in the test.

#### 7.2.1.6 Limitations

For some endpoints and functionalities such as searching by keywords, due to the usage of language processing(NLP), the key term may not appear in some result articles. Therefore,

there is no efficient way to test this in API testing. However, the correctness should be mostly tested in the NLP testing.

Also when the API is listing the results, the result set is sorted by the term frequency beforehand. However, the validity of this is not tested as well.

Because for the searching of location, the keyword and the result are not necessarily exactly matching as well, the fuzzy comparison is used in these cases as an alternative way. For example, keywords and results may be compared case-insensitively or we may only require a keyword to be the substring of the result.

Since we are using the public platform for the database, most of the functions are completed so that it could automatically throw exception errors. So the tests are quite simple that could not cover any other unexpected cases.

#### 7.2.1.7 Improvement

In the process of testing, some potential bugs in the API have been found and fixed. The test script could also possibly be strengthened by introducing randomize tests, which means generating some random input to test whether the code satisfies the requirements. However, due to time limitation, this has not been implemented with the current test suite.

### 7.2.2 Scraper Component

#### 7.2.2.1 Environment

- Python 3.7
- Scrapy
- BeautifulSoup

#### 7.2.2.2 Overview of Test Cases

All of the test cases are used to check the correctness of data preprocessing of the scraper. Given valid HTML files or URLs, the scraper is required to scrape the correct websites, and the extracting data is needed to be complete and accurate.

For testing the correctness of data preprocessing of the scraper, we provided a consistent valid HTML file and consistent URLs. By using the given files, we test functional correctness of these functions by comparing the function output with the correct filtered data which is manually filtered. Also, the data source website sometimes returns invalid HTML, to avoid this we also provided some most frequently-seen invalid HTML to see if data preprocessing functions can ignore these HTML.

The scraper can also auto-update the newer data into the database and filter out the old-time data. To test this functionality, we always auto-generate newer and old-time data to check the functional correctness.

### 7.2.2.3 Details of Test Cases

Test case ( <b>test_scrapy.py</b> )	Explanation
test_empty_last_activity()	To check if the scraper returns a valid HTML file to extract data
test_last_activity_basic_attrs() test_LA_get_posts() test_get_source_text_for_onepost()	To check the data preprocessing of the scraper with a valid HTML file. To ensure that given a valid HTML file, the data preprocessing can extract the accurate and complete data
test_filter_get_text_byp_invalid_url() test_invalid_post() test_invalid_source_content()	To check the data preprocessing of the scraper with an invalid HTML file. To ensure the scraper is smart enough to ignore invalid input.
test_update_post()	To test the auto-update ability of scraper. We always auto-generate some newer and old-time data as input, to check if the update function is able to pick the newer data and filter out old-time data

#### Pytest Result:

```

===== test session starts =====
platform win32 -- Python 3.7.4, pytest-5.2.1, py-1.8.0, pluggy-0.13.0
rootdir: E:\code\SENG3011_APIInteresting\PHASE_1\TestScripts
plugins: arraydiff-0.3, doctestplus-0.4.0, openfiles-0.4.0, remotedata-0.3.2
collected 8 items

test_scrapy.py ..... [100%]

===== warnings summary =====
C:\Users\c1c87\Anaconda3\lib\site-packages\html5lib\tree\base.py:3
  C:\Users\c1c87\Anaconda3\lib\site-packages\html5lib\tree\base.py:3: DeprecationWarning: Using or importing the ABCs
from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working
    from collections import Mapping

C:\Users\c1c87\Anaconda3\lib\site-packages\win32\lib\pywintypes.py:2
  C:\Users\c1c87\Anaconda3\lib\site-packages\win32\lib\pywintypes.py:2: DeprecationWarning: the imp module is deprecated
in favour of importlib; see the module's documentation for alternative uses
    import imp, sys, os

test_scrapy.py::test_get_source_text_for_onepost
  C:\Users\c1c87\Anaconda3\lib\site-packages\scrapy\core\downloader\webclient.py:4: DeprecationWarning: twisted.web.clie
nt.HTTPClientFactory was deprecated in Twisted 16.7.0: please use https://pypi.org/project/treq/ or twisted.web.client.A
gent instead
    from twisted.web.client import HTTPClientFactory

test_scrapy.py::test_get_source_text_for_onepost
  C:\Users\c1c87\Anaconda3\lib\site-packages\scrapy\core\downloader\contextfactory.py:53: DeprecationWarning: Passing me
thod to twisted.internet.ssl.CertificateOptions was deprecated in Twisted 17.1.0. Please use a combination of insecurely
LowerMinimumTo, raiseMinimumTo, and lowerMaximumSecurityTo instead, as Twisted will correctly configure the method.
    acceptableCiphers=self.tls_ciphers)

test_scrapy.py::test_invalid_post
  C:\Users\c1c87\Anaconda3\lib\site-packages\bs4\_init_.py:294: UserWarning: "b'scrappy_test_data/invalid_post.html'" 1
ooks like a filename, not markup. You should probably open this file and pass the filehandle into BeautifulSoup.
    BeautifulSoup. % markup)

-- Docs: https://docs.pytest.org/en/latest/warnings.html
===== 8 passed, 5 warnings in 20.94s =====

```

#### 7.2.2.4 Coverage

The test script achieves function coverage by testing all the core functions in scraper. Some of the functions are not tested since they are just used to call the core functions for combining functionalities together.

The branch coverage is also achieved by providing valid/invalid HTML files and URLs to allow all of the branches in each function to be run, and then we test the branch output with the expected output to check the correctness.

#### 7.2.2.5 Limitations

Since the data source website always returns variable URLs and HTML, we cannot test the scraper with all possible input. And the scraper is only able to scrape one HTML file at each script call in our implementation, so it is impossible to scrape multiple HTML in one test script. Therefore, we have to manually test if the scraper can scrape multiple HTML pages.

#### 7.2.2.6 Improvement

During the test, we noticed that we didn't have a method to check if the HTML file is valid. This is important since all the previous scraper codes are run under the assumption that all input HTML files are valid. And the update functionality needs to compare the latest date of our database with the date of new coming data, since the latest date of database update periodically, so we modify the test script and now it is able to auto-generate newer and old-time data to ensure the test covers both of the situations.

### 7.2.3 Data Processing (NLP) Component

#### 7.2.3.1 Environment

- Python 3.7
- pytest

#### 7.2.3.2 Overview of Test Cases

Data processing component contains four minor components, `Date_formatter`, `Location_checker` and `Geocode_Locaion` will be imported in `NLP_Processor`. Each component will be tested.

`Date_formatter` needs to extract date and time from an extracted pattern from spaCy for multiple times in for an article and return a period of time or an exact time in the desired format. `Date_formatter` will firstly be tested against different date patterns (e.g. 2020/12/12), date range patterns (e.g. Feb 3-5), time patterns (e.g. 12:15 pm) to examine its correctness for a single pattern, then be tested against multiple patterns to examine its ability to analyze the date and time patterns in order to return event date or event date period.

Location\_checker and Geocode\_Locaion will be tested together. They will be tested against a bunch of location-like patterns including countries' general name, countries' short name, cites' name and non-location name. They need to capture the names of non-duplicate locations and transfer them into google place id.

The above minor components will then combine in NLP\_Processer for overall testing against some short articles, the output report, location\_keywords, keyword\_frequency\_list will be examined to check correctness. The input test data will include the main text and the publication date of articles. Then some longer articles which contain multiple cases will be tested to examine NLP\_Processer's ability in separating reports.

### 7.2.3.3 Coverage

Date\_formatter, Location\_checker and Geocode\_Locaion will be tested individually with specific test data. Then they will be combined and tested with NLP\_Processer to check the overall performance. All known potential failures will be tested and all functions will be tested.

### 7.2.3.4 Details of Test Cases

Test case ( <b>test_date_formatter.py</b> )	Explanation
test_single_date_normal_format() test_single_date_riched_format()	To check the ability in converting exact date string into correct format
test_date_period()	To check the ability in converting ranged date string into correct format
test_time()	To test the ability in converting time string into correct format
test_return_exact_date_1() test_return_exact_date_2() test_return_date_range_1() test_return_date_range_2() test_return_date_range_3() test_return_date_add_time_1() test_return_date_add_time_2()	To processing a series of date strings, time strings and return event date string
test_error_date_ignore() test_error_time_ignore()	To test the ability in distinguishing valid or invalid patterns

#### Pytest Result:

```
(base) PS C:\Users\ASUS\se3011\SENG3011_APIInteresting\PHASE_1\TestScripts> pytest .\test_date_formatter.py
===== test session starts =====
platform win32 -- Python 3.7.4, pytest-5.3.5, py-1.8.1, pluggy-0.12.0
rootdir: C:\Users\ASUS\se3011\SENG3011_APIInteresting\PHASE_1\TestScripts
collected 12 items

test_date_formatter.py ..... [100%]

===== 12 passed in 0.07s =====
```



Test case ( <b>test_location_related.py</b> )	Explanation
test_location_checker()	To test the ability to separate location names and country names
test_geocode_location() test_duplicate_locations()	To check the ability in converting location string to place id
<b>Pytest Result:</b> <pre> ===== test session starts ===== platform win32 -- Python 3.7.4, pytest-5.3.5, py-1.8.1, pluggy-0.12.0 rootdir: C:\Users\ASUS\se3011\SENG3011_APIInteresting\PHASE_1\TestScripts collected 3 items  test_location_related.py ... [100%]  ===== 3 passed in 3.13s ===== (base) PS C:\Users\ASUS\se3011\SENG3011_APIInteresting\PHASE_1\TestScripts&gt; </pre>	

Test case ( <b>test_NLP.py</b> )	Explanation
test_short_article() test_short_article_multiple_reports() test_long_article_single_report() test_comprehensive_report_1() test_comprehensive_report_2() test_comprehensive_report_3()	Overall testing cases. Inputs are main text of a news, outputs are a list of reports in correct format
<b>Pytest Result:</b> <pre> (base) PS C:\Users\ASUS\se3011\SENG3011_APIInteresting\PHASE_1\TestScripts&gt; pytest .\test_NLP.py ===== test session starts ===== platform win32 -- Python 3.7.4, pytest-5.3.5, py-1.8.1, pluggy-0.12.0 rootdir: C:\Users\ASUS\se3011\SENG3011_APIInteresting\PHASE_1\TestScripts collected 6 items  test_NLP.py ..... [100%]  ===== warnings summary ===== C:\Users\ASUS\Miniconda3\lib\site-packages\plac_ext.py:6   C:\Users\ASUS\Miniconda3\lib\site-packages\plac_ext.py:6: DeprecationWarning: the imp module is deprecated in favour   of importlib; see the module's documentation for alternative uses     import imp  test_NLP.py::test_short_article test_NLP.py::test_short_article_multiple_reports test_NLP.py::test_long_article_single_report test_NLP.py::test_comprehensive_report_1 test_NLP.py::test_comprehensive_report_2 test_NLP.py::test_comprehensive_report_3   C:\Users\ASUS\Miniconda3\Scripts\pytest-script.py:10: DeprecationWarning: [W010] As of v2.1.0, the PhraseMatcher does   not have a phrase length limit anymore, so the max_length argument is now deprecated.     sys.exit(main())  -- Docs: https://docs.pytest.org/en/latest/warnings.html ===== 6 passed, 7 warnings in 20.70s ===== </pre>	

### 7.2.3.5 Limitations

We can only test cases that we can imagine or those we have encountered. Therefore we cannot assure all aspects are tested. We do not know if we catch all error patterns when processing. For example, in testing the Date\_formatter, we can only test the format of date or time that we have encountered and eliminated incorrect captured patterns that we have seen. Furthermore, the amount of test input is relatively small for NLP\_Processer. As we need to check the input manually to decide the correct output

### 7.2.3.6 Improvement

During testing, we captured some incorrect patterns which should not bypass the filter, so we updated the filter several times to reinforce its ability to distinguish valid and invalid patterns. We also observed duplicated location names in a single report thus fix a logic bug in Geocode\_Locaion component. By observing inputs and outputs from NLP\_Processer, we can update the vocabulary of NLP\_Processer thus it can recognize and capture more diseases and syndromes.

## 8. Group & Project Management

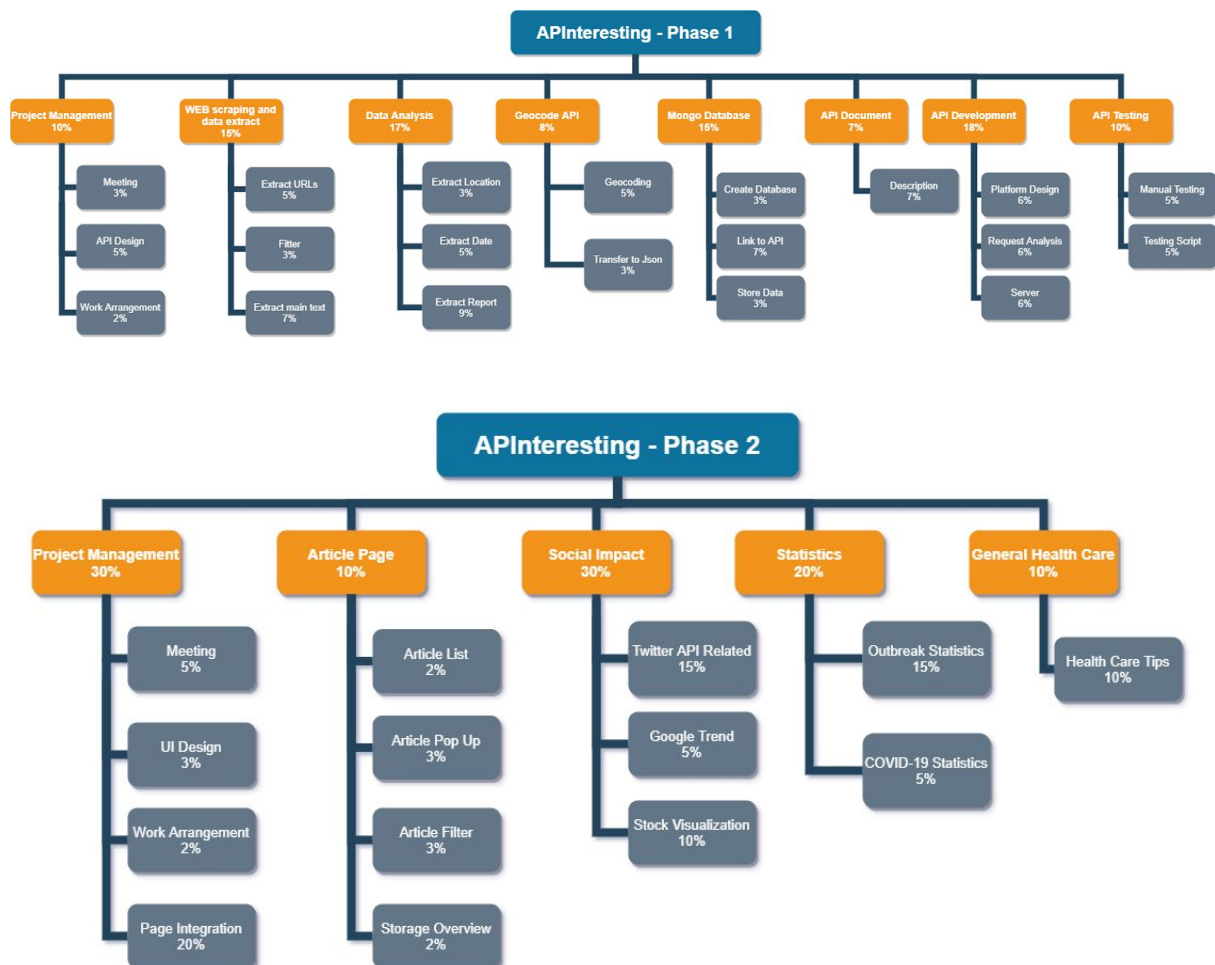
### 8.1 Work Distribution

<b>Task(s)</b>	<b>Member(s)</b>
Design Report	Everyone
API Development	Webster, Ping
API Document	Everyone
API Testing	Webster
Firebase	Ping, Ryan
Web Scraping & Data Extract	Gary
Text Mining	Yahnis
NLP Testing	Yahnis
Geocode API	Yahnis
Data Analysis	Yahnis, Gary
Front-end Design	Everyone
Front-end Skeleton	Ping
Article List Page	Gary, Webster
Social Impact - Stock	Webster
Social Impact - Google Trend	Gary, Webster
Social Impact - Twitter	Ping, Yahnis
Statistics - Outbreak Statistics	Yahnis
Statistics - COVID-19 Statistics	Gary, Webster
Healthcare Page	Ryan
Page Integration & UI Adjustment	Webster, Ryan
Final Report	Everyone

## 8.2 Grantt Chart

Task	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
<b>Phase 1</b>										
Design Report										
Web Scrapping										
Text Analysis										
Google Geocode API										
Firebase Database										
API Development										
API Documentation										
API Testing										
<b>Phase 2</b>										
Page Skeleton										
Article List Page										
Social Impact Page										
Statistics Page										
Healcare Page										
Page Integration										
Final Report										

## 8.3 WBS Diagrams



## 9. Evaluations

### 9.1 Requirements Status Table

Requirements	Status
<b>Must-have</b>	
As a medical officer, I want to be able to learn about specific diseases from articles on different websites so that I can know how severe the outbreak is and stay informed	Implemented
As a data scientist, I want to be able to know the location for different disease outbreaks over a period of time so that I can analyse the severity of different areas	Implemented
As a medical officer, I want to be able to monitor the severity of outbreaks on statistics in different places in the world so that I can adjust the policy to protect the public	Implemented
As a data analyst, I want to be able to obtain some meta-information for an article without actually reading the content so that it is easier for me to handle the article data	Implemented
<b>Should-have</b>	
As a data scientist, I want to be able to see what the public is searching for an outbreak	Implemented
As a data scientist, I want to be able to see how social media reacts to a disease outbreak	Implemented
As a public, I want to be able to find out advice about how I can protect myself from a disease	Implemented
<b>Could-have</b>	
As a data scientist, I want to be able to see if there is a relationship between the stock market and an outbreak	Implemented
As a data scientist, I want to be able to see if there is a relationship between the currencies and an outbreak	Not Implemented
As a data scientist, I want to be able to see if there is a relationship between the consumer price and an outbreak	Not Implemented

API Related	
As a user of API, I want to get a list of article overviews over specific criteria so that I can demonstrate a preview of them in my application	Implemented
As a user of API, I want to get a list of articles in detail over specific restrictions so that I can display them on my application	Implemented
As a user of API, I want to get an article in detail by ID so that I can display them in my application	Implemented
As an user of API, I want to get analyzed reports from an specific article so that I can perform analysis on it	Implemented

## 9.2 Key Achievements Summary

In the project, these important benefits have been achieved.

Backend:

- The API is flexible to be used with manageable code conforming high cohesion and low coupling requirement.
- The quality of result that the API returns is relatively high as the extraction of meta-data and the optimization made on searching.
- Integrated data from multiple data sources by cleansing, and visualizing, which increases the value of the data regarding academic or researching.
- In the text mining (NLP) part, we are able to geocode the address of the locations found in the articles and assign them full address name with city and country using Google Geocoding API.
- The NLP is able to distinguish how many different reports an article has instead put all found information into one report. This is achieved by mapping of keywords and examining their correlationship.

Frontend:

- All charts, graphs and data can be exported easily for the user to perform further analysis or quote directly.
- Ability in examining the social and economic impact of a pandemic, current popular topics related to diseases by analyzing twitter, Google's data.
- A vast amount of interactive and intuitive charts to display data in an elegant manner

## 9.3 Wished Prerequisites Skills

- The ability to handle and process a large amount of data in a short time
- The ability to decompose a complex system to a smaller scale of code
- More knowledge of web application development
- The ability to build and train a neural network model for predicting disease amount.
- The ability in machine learning to provide better NLP experience

## 9.4 Challenges and Decisions

During the project, we met these challenges and made these decisions.

- NLP for generating multiple reports from an article is hard as the phase matcher will do things in a linear manner and it has no knowledge about disease. We create a keyword correlation lookup table to help the NLP processor separate generated data in multiple reports if required.
- During the testing of the backend, the daily quota for Google Firebase exceeded due to the frequent refreshing and requesting. Thanks to the flexibility of the backend structure, we were able to switch back to a dummy data generator temporarily for the day.
- When we try to obtain Google Trends data for the social impact components, we found Google does not provide raw data for Google Trends. Seeking a compromise, we implemented this functionality by embedding Google Trends just like a widget.
- To display the stock indexes in the same graph with the number of the confirmed cases of COVID-19, we realized the number of confirmed cases is too large to be displayed, we managed to use multiple axes and scales to display the confirmed cases.
- To display a map with extensive labels of outbreak locations have a lot of hidden issues. Firstly, Google will only accept static map requests for less than 15 labels if they are not geocoded. Thus I need to geocode all my locations in advance. As everything is in a map function and this creates an async function (Promise) within another async function (Promise) and this generates inconsistency issue and It is hard to debug. This is finally solved by trial-and-error. Moreover, Google will reject geocoding requests if it hits the limitation request number per second. This is solved by using a math compression function to reduce the number of requests meanwhile keep the quality.

## 9.5 Further Improvements

This quality of the project can be improved by these possible improvements below.

- Use MongoDB or MySQL as the database for backend data storage. Currently considering the scale of the project, Google Firebase has been used as the database. It is light-weighted, easy to construct codes but lacks advanced functionalities.
- At NLP part, the processor can be implemented as a trained machine learning module that will constantly improve itself automatically instead of importing pre-exist dictionary data.
- NLP processors can only separate diseases and syndromes in multiple reports, not location and date because phase matcher will only handle in linear order the relationship of date with keywords in time scale is unknown. If we were aiming for multiple reports, not a single report at the first, another way for NLP may be used thus the location and date can be separated into multiple reports.
- Establish a consistent style for the front-end. We had done the front-end page with excessive libraries because of the lack of such an agreement on the library, in the process of front-end implementation, members tend to import more libraries instead of sticking to those that exist.
- Most graph components can be improved by implementing more user-customisable functionalities like filtering or searching.
- Adding more functionality to our social impact analysis on the platform. Users could be able to define search terms dynamically for google trends and possible data processing techniques could be used on the tweets we extract from the Twitter API
- The API could provide more endpoints for various features, like getting the latest epidemic disease. We have very comprehensive data in our database but just convert them into relatively plain features.
- There should be a comparison function for different countries in the outbreak statistics page, we do not have enough time so we quit implementing this nice feature.