SEMESTER 2 2019

MASHUP/DOCKER REPORT

CAB432: CLOUD COMPUTING

INDIVIDUAL TASK – ASSESSMENT 1

KEVIN DUONG – N9934731

QUEENSLAND UNIVERSITY OF TECHNOLOGY

Contents

Introduction

WordReport is a web mashup that utilises News and Keyword Extraction APIs, providing users current news from around the world by showcasing trending news headlines via key words that many news sources are covering in recent times. Important 'keywords' and 'phrases' are scanned from a list of URLs, and outputs 10 words compiled by the APIs. Clicking one of the words will prompt up all news articles relating to the keyword. If the user wants to read the full news, then clicking one of the headlines/images will show the article's website.

Users must first search a country that is ***available*** in the search bar in which the auto suggestion will provide countries that can be looked up. After clicking search, the APIs will begin fetching URLs and words related to that country and then direct the user to another page with the returned results. A table of 10 words will appear on this webpage, and users can either lookup another 10 words from a different country or click one of the words to preview its news articles. At this stage, the user is directed to the final point of the website and can preview the headlines, sources, descriptions, images, and links of each article before dwelling into the source's website for more information. See Appendices A and B for a full walkthrough.

News API

https://newsapi.org/docs/client-libraries/node-js

News is a simple HTTP REST API for searching and retrieving live articles from all over the web. Various methods of calling news includes keywords and phrases, the date it was published, sources, and news from different countries.

The API contains three endpoints which provides 'top headlines' about breaking news for any country, 'everything' such as finding queries about a word, sorting articles or adjust the number of articles, and 'sources' to handle news publishers and language/country preferences.

Keyword Extraction API

https://rapidapi.com/UnFound/api/keywords-extraction2

Keyword Extraction can extract the most important 'Keywords' and 'Phrases' from the text or URL that being given. It can extract entities such as names of important People, Organizations, Places, and even Concepts, Sub-topics, etc.

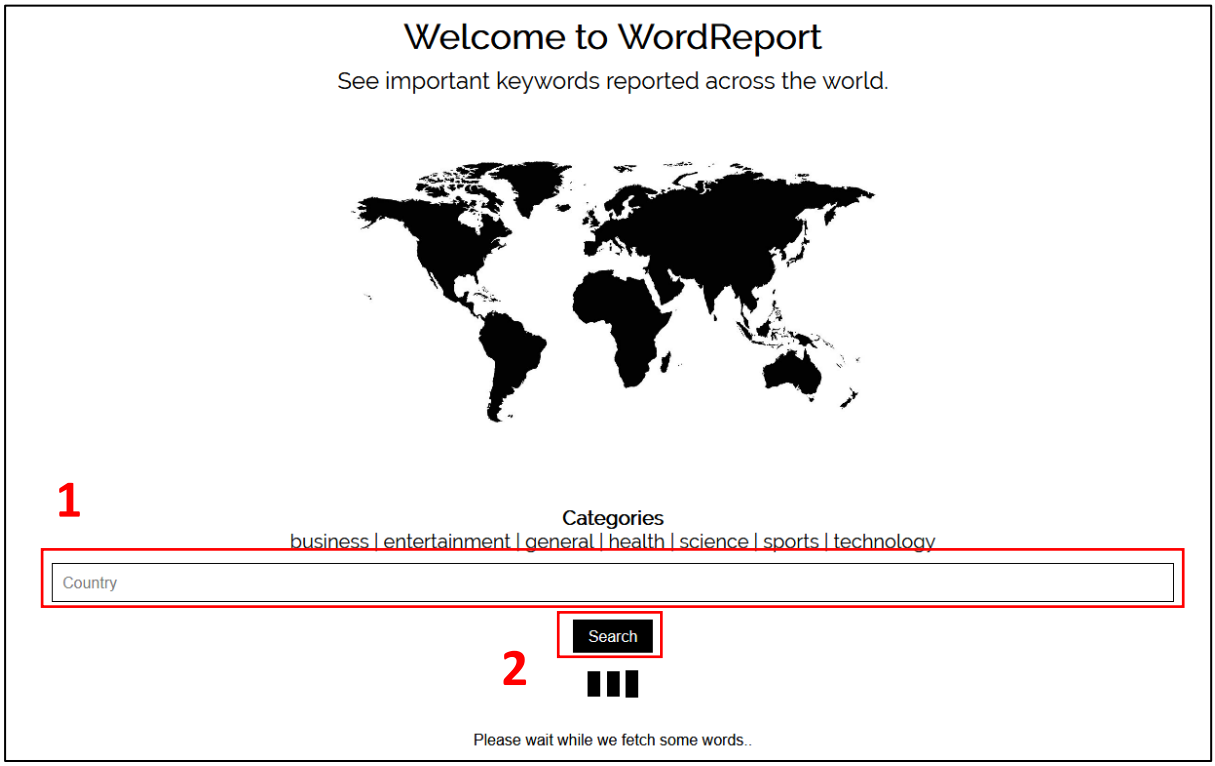
The API can take multiple text or URLs as input and can specify the number of keywords that needs to be extracted for results. Inputting URLs can correspond to news, blogs, and other webpages across the internet.

Mashup

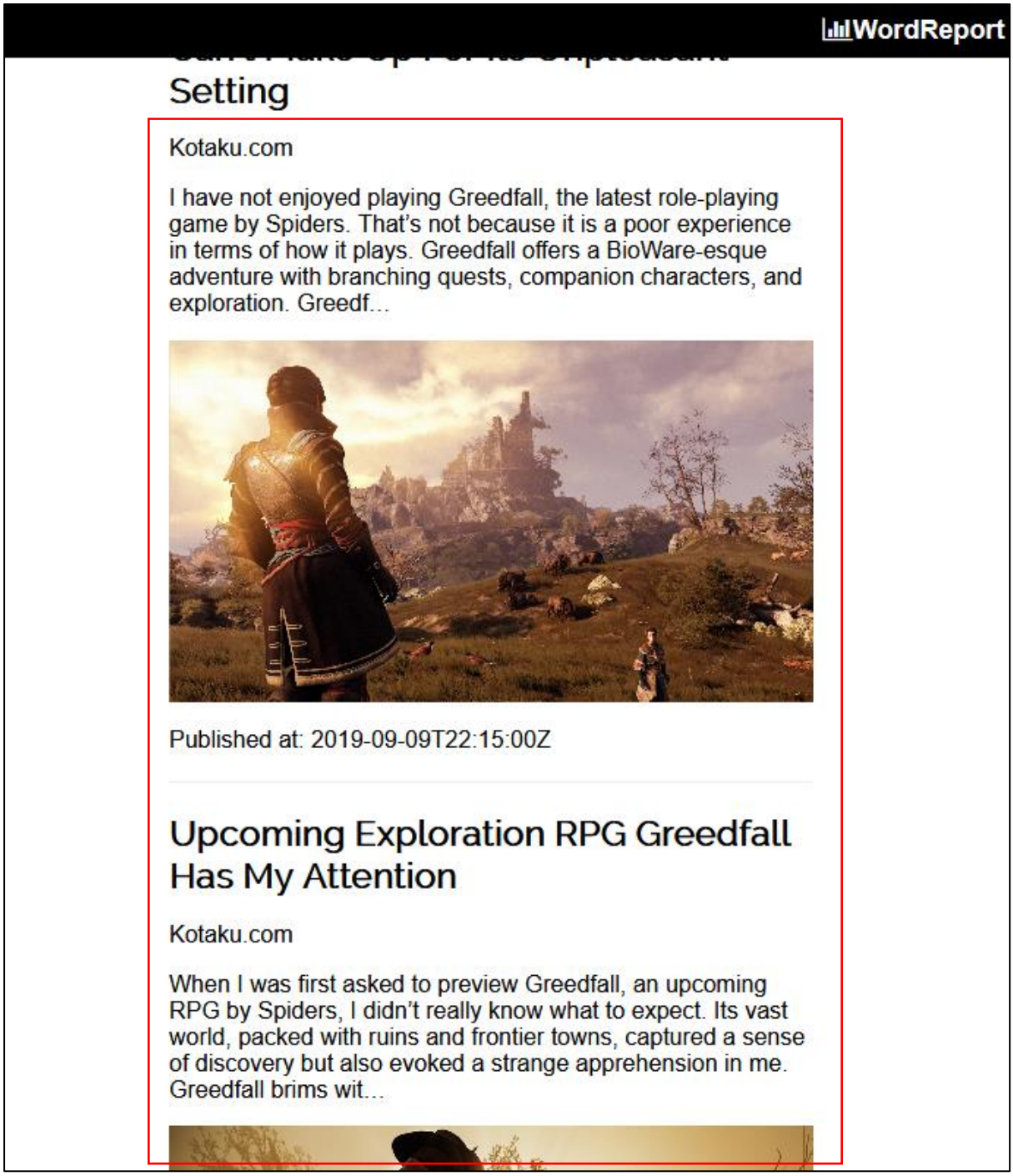
Use Case A

As a user, I want to find out what's trending right now, so I can get onboard with popular news.

The user is welcomed to the index page and begins to lookup what Australia has to offer by typing away in the search box and clicking the *search* button (the query will be abbreviated to its country code as a reference). This gives them time to wait for the keywords to be fetched.



The user is then presented with ten words after waiting for the data to be loaded. Here, the user is interested in the word 'greedfall' being #1 and continues from there, only to see news articles about Greedfall. To their surprise, they learn that Greedfall is a new upcoming video game.

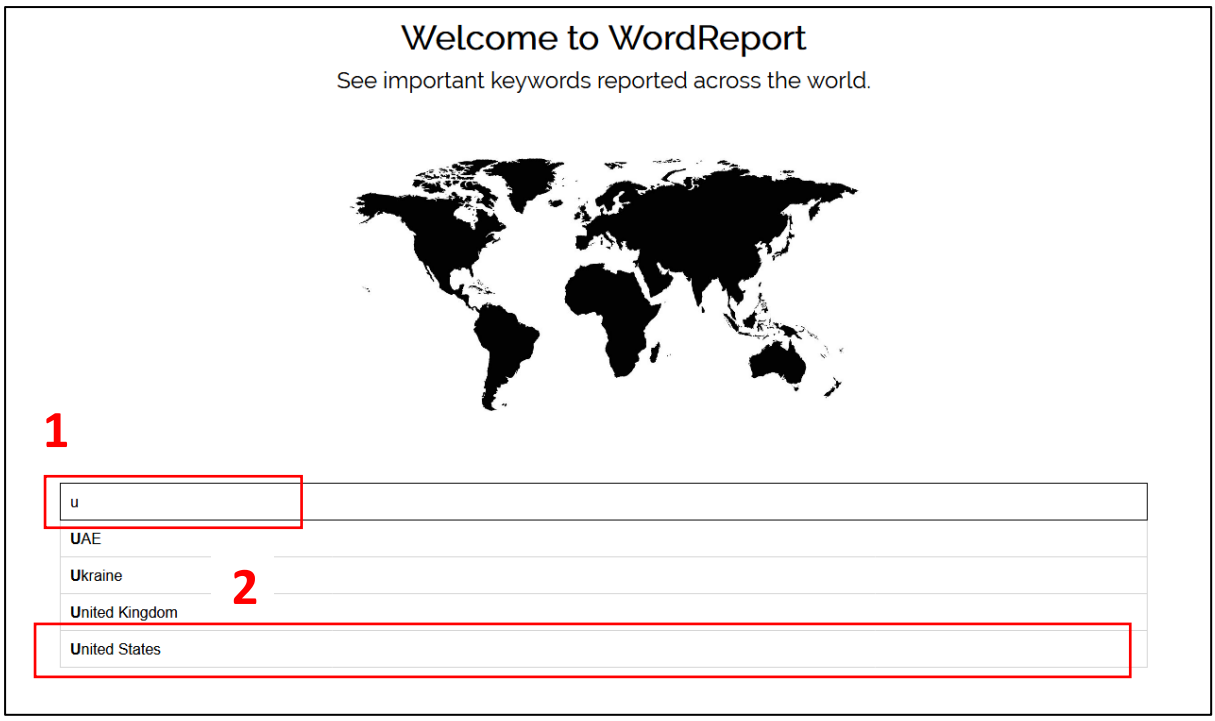


APIs utilised: Keyword Extraction, News

Use Case B

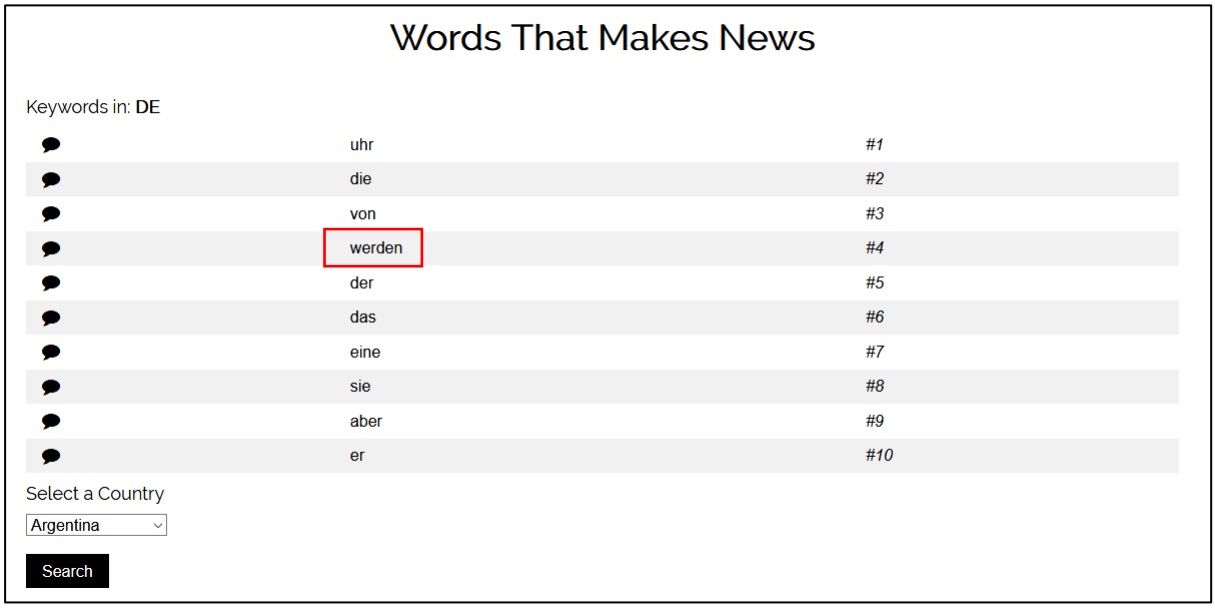
As a reader, I want to know what's trending in another country, so I can get an idea of what's happening there.

The user begins on the index page and starts searching right away in the search input. They suggest looking up trending keywords from 'United States' to see what people are talking about over there.



As a result, they find out that many keywords do in fact relate to what's happening around that country. The user then decides to lookup another country, this time in Germany.

After loading, the keywords retrieved have appeared to be shown in deutsche, reflecting the emphasis of the use of the language in Germany. Clicking on the word ‘werden’ has shown a degree of reference from sources and headlines in the country.

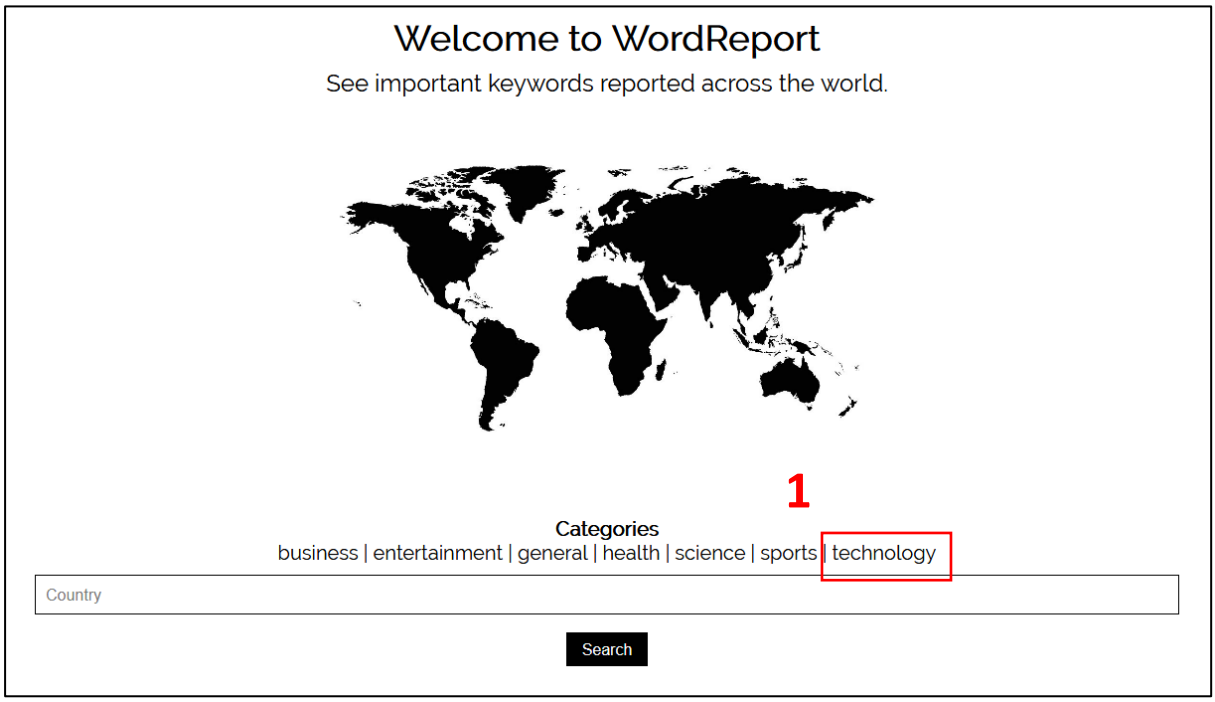


APIs utilised: Keyword Extraction, News

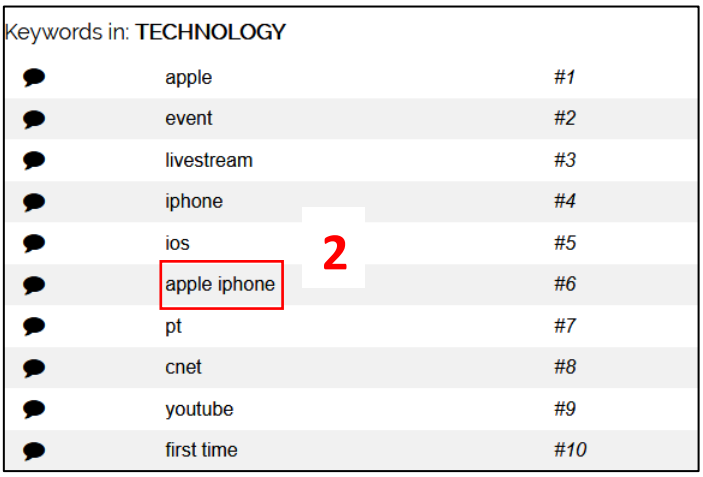
Use Case C

As an enthusiast, I'd like to see what's trending in technology, so I can keep up with the subject.

The user has the option to lookup 'technology' on the index page labelled under the category section. They then click and wait for their request to be completed.



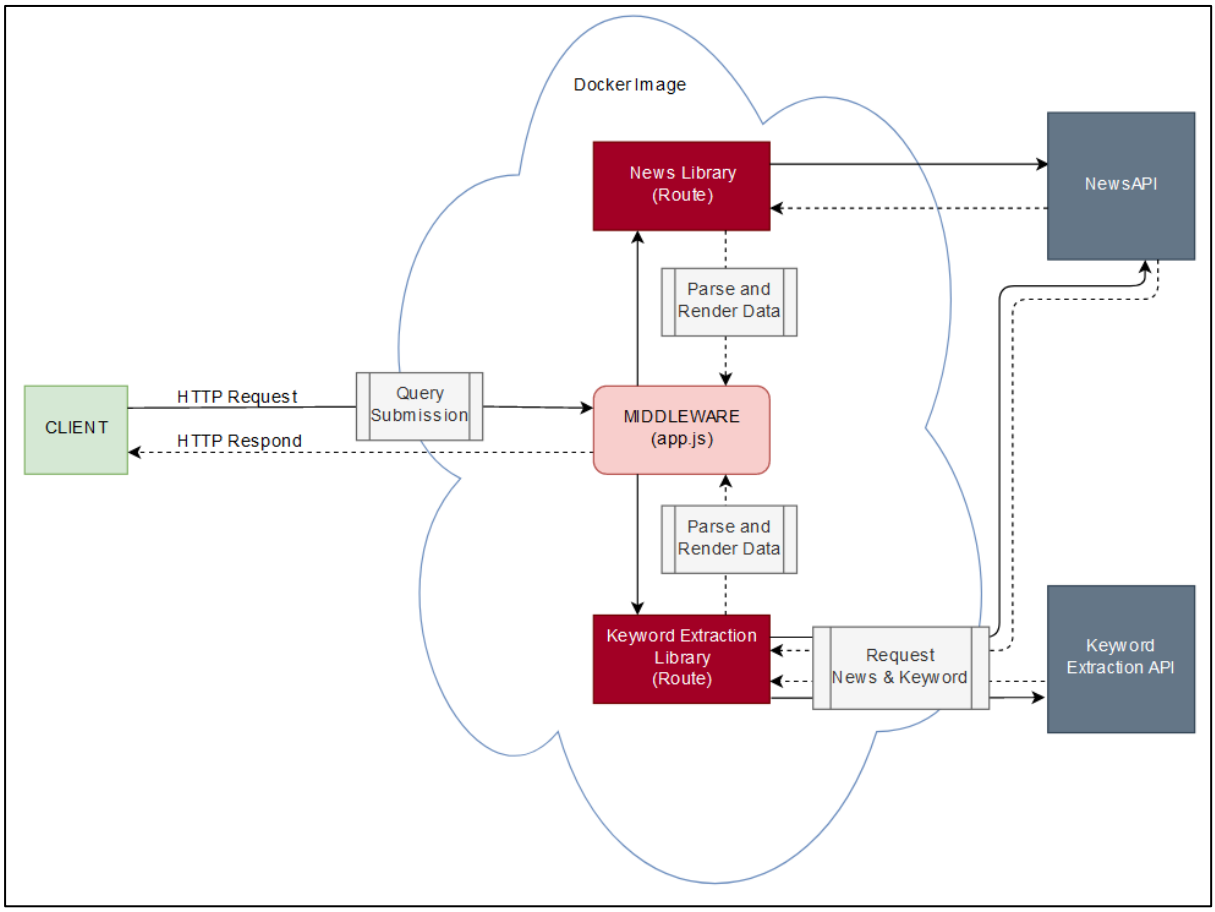
By looking at the important keywords, the user decides to click and read the latest news about Apple and their latest iPhone.



APIs utilised: Keyword Extraction, News

Application Description

Overall the architecture of the application shows a great deal of processing data in the server-side than the client-side. Requests by the client are passed through to the middleware which is also linked to two other routes: the news and keyword extraction libraries. These libraries heavily refer to the API services used to make this application work. The APIs respond, with filtering provided through the libraries so that correct data is sent to the client's needs in the client side.



Client Side

The client will often post data to request information whether it's typing a search query or clicking a link. All information is sent through the server side in HTML form POSTs or hypertext references. Validation is used to ensure that proper POSTS are made when searching a dynamic query. If approved, the information will then be processed and respond back to the user by directing them to a new page. Subsequently, the user will be able to see a new display of elements and feedback from their requested data that they inputted.

Server Side

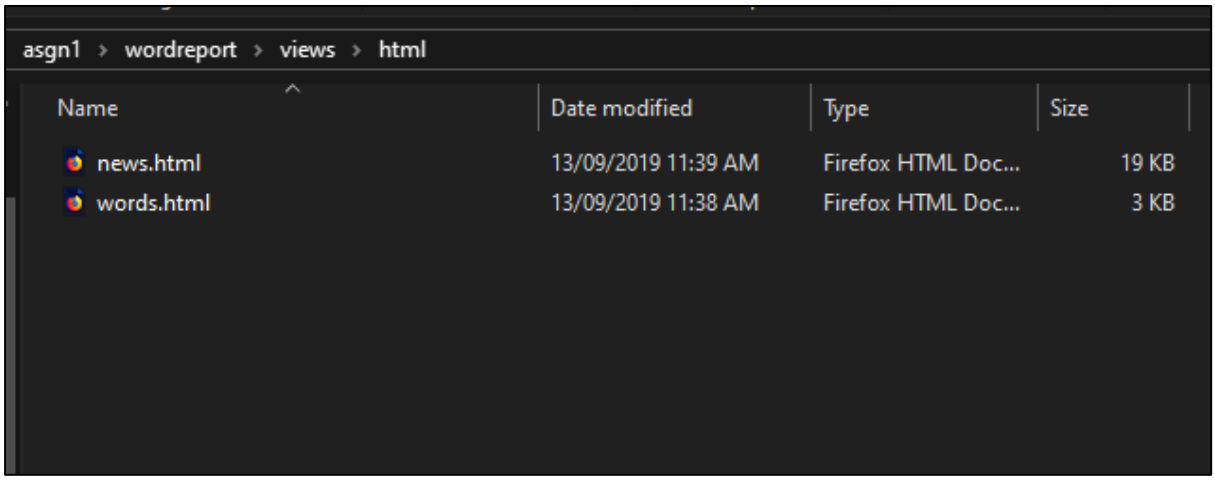
Beyond the client lies the server side which handles any request that the client requires, by setting up a middleware server to take HTTP requests from the client and responses from routes. Furthermore, the routes themselves are created as libraries to request data from API services, which benefit the client's needs by providing proper information to the server. While the *News* library solely requests information from the News API, the *Keyword Extraction* library needs to request both APIs to properly serve its response data and forward it to the middleware, which now has to respond to the client. The Keyword Extraction API requests 20 news URLS from the NewsAPI, given that the NewsAPI requests a

query for either country/category. It then extracts 10 important keywords from the URLs and then parses the words into a proper display for the webpage.

Parsing and Rendering Data

When data is given to the libraries from the APIs, they must only extract relevant information that the client wishes to see. Each data is composed as a JSON object that can be parsed and extracted in HTML. The filtered data can now be easily sent through the server and render to the client side without having to compile unnecessary information. They are then written as a HTML file to load in and render in the webpage upon server requests and responses.

```
55  function parseNews(data) {
56
57      let s = '';
58      let news = data.articles;
59
60      for (let i in news) {
61          s += `
62                  <h3><a href = "${news[i].url}"><b>${news[i].title}</b></a></h3>
63                  <p>${news[i].source.name}</p>
64                  <p>${news[i].description}</p>
65                  <div class = "container">
66                      <a href = "${news[i].url}"> <img alt = "${news[i].title}" src = "${news[i].urlToImage}"/></a>
67                  </div>
68                  <p>Published at: ${news[i].publishedAt}</p>
69                  <hr>
70              `;
71      }
72
73      // Write data for HTML.
74      fs.writeFile('./views/html/news.html', s, (err) => {
75
76          // In case of a error throw err.
77          if (err) throw err;
78      })
79
80      return s;
```



Pug html

An alternative solution for building webpages compared to standard HTML or PHP, Pug is a HMTL templating engine that provides useful features to render and implement data from JSON or other static files, and can organise templates for much more simple and flexible ways to work with content efficiently. Pug is mainly used for server-side responses and is compiled into HTML code once it is sent to the client side. Since I used a function to parse and write HTML files for rendering, Pug lets me include this and any other static files, whether it's HTML, CSS or JavaScript. Overall, all webpages are written in pug.

Issues encountered

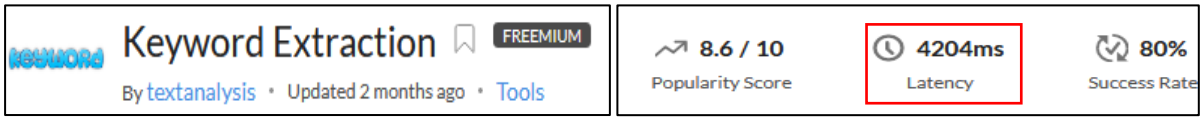
Request Problems

Almost all countries available in the database can be searched, however some countries produce errors while trying to request and compile data. One particular example is looking up China (cn) to find its keywords, only to realise JSON is unable to parse non-Unicode characters. Unlike most countries reporting their news, China only uses Chinese characters for their writing format which makes it difficult to work with JSON. The *Keyword Extraction API* is also unable to resolve this issue, which leads to a compromise of removing China as an option to look up its keywords and news. A test plan for this issue is mentioned later in the report.

```
SyntaxError: Unexpected end of JSON input
    at JSON.parse (<anonymous>)
    at IncomingMessage.<anonymous> (C:\Users\Kevin\Desktop\asgn1\wordreport\routes\words.js
:69:32)
    at IncomingMessage.emit (events.js:203:15)
    at endReadableNT (_stream_readable.js:1145:12)
    at process._tickCallback (internal/process/next_tick.js:63:19)
```

API Latency

While inevitable, loading times for requesting queries can take some time before data responds back to the client. The Keyword Extraction API can take up from 2874ms to at least 4204ms maximum if internet ping worsens. This leads to having to wait around 30-40 seconds for all countries/categories to produce results for the user. If the user is experiencing a slow connection, then requesting a query may cause issues to the server side, as it is necessary for the NewsAPI to fetch the URLs in time to write the file and pass it on to the Keyword Extraction API for request.



Use of Docker

A Dockerfile is a text document that includes line commands to assemble an image in Docker. The file is placed next to the web mashup folder which is tasked to copy the source directory and build an image based on the web application. This means it will contain all files and folders of the original source performed in line 2.

```
1  FROM node:buster-slim
2  COPY /wordreport .
3  EXPOSE 80
4  RUN npm install
5  CMD ["node", "app.js"]
```

For the first line in Dockerfile we establish the use of Node.js's buster-slim, a lite version of the recommended 10.16.3 LTS. This allows us to build a smaller image, whereas using the default 'node:10' will give us an image size of 920MB. Buster-slim provides a lightweight, yet faster build of an image with all the dependencies carried with it. Port 80 is exposed in line 3 for our server (app.js), and 'RUN npm install' is given in line 4 to install our dependencies. Finally, in line 5, we give the command to run the server. The image is then pushed to my Dockerhub account, ready to be pulled back later on to the AWS public VM (Appendix C).

Testing and Limitations

A basic test plan table is created to ensure the web application is behaving as it should be. Each test looks at an ordinary task where a user would interact and should produce an outcome that is ideal to how the web application reacts. A result column is also added to determine whether the test passes or fails, and references to the tasks can be seen in the screenshot guide section (Appendix B).

~ RH = Response Handling
~ cn = China

Task	Expected Outcome	Result	Screenshot Guide (Appendix B)
Search for words (country)	Displays 10 keywords related to country	PASS	01
Search for words (category)	Displays 10 keywords related to category	PASS	02
Enter incorrect search	Prevents submission	PASS	03
Use dropdown list to search	Results display 10 new keywords	PASS	04
Loading interaction	Shows loading animation while waiting	PASS	05
Click keyword for news	Directs to a new page full of news	PASS	06
Click article to read full news	Redirects to the source article	PASS	07
RH: search for invalid country (URL)	Should reject in URL parameter (local): http://127.0.0.1:3000/words/**cn**	PASS	08
RH: search for incorrect category (URL)	Produce an empty result page	PASS	09
Fetch keywords at a slow connection	Display a slow connection message	PASS	10

Overall there are limitations that are added to mitigate errors from the user. These are usually handled on an error page indicating the user that they have submitted an error or that there is a problem with the request from the application. Another error handling function is the client-side validation, which upon query searches, users must fill out queries that are available in the auto suggestion list and also not to leave a blank on the search bar if they're trying to find keywords from a country.

Extensions

Third API

Due to the time constraint and the need to complete the basics of the application, a third API was not considered. A proposed idea was adding a google maps API that would allow users to explore the world and click on a country to search for trending keywords. This implementation would overhaul my current mashup and explore new features suitable for more use cases. Another suggestion was adding Twitter to read people's thoughts and opinions on the latest news and keywords.

Filter Preferences

Prior to searching for keywords, it is preferable to suggest filters for producing more accurate keywords that are desirable. For example, adding a data range when the word is used a lot, or include a sort field via relevancy or popularity of the word.

Email Notifications

A feature that would serve well for users would be the idea of subscribing to a keyword that they'd like to read more about for future references of the term. Email notifications would provide the user news articles relating to the keyword just like how news articles are extracted using the query parameter in the mashup application.

References

1. NewsAPI. (n.d). Documentation. Retrieved from: https://newsapi.org/docs/client-libraries/node-js

2. UnFound. (2019). Keywords Extraction API Documentation. Retrieved from: https://rapidapi.com/UnFound/api/keywords-extraction2

3. Docker. (n.d). Dockerfile reference. Retrieved from: https://docs.docker.com/engine/reference/builder/

Appendix A – User Guide

Getting Started

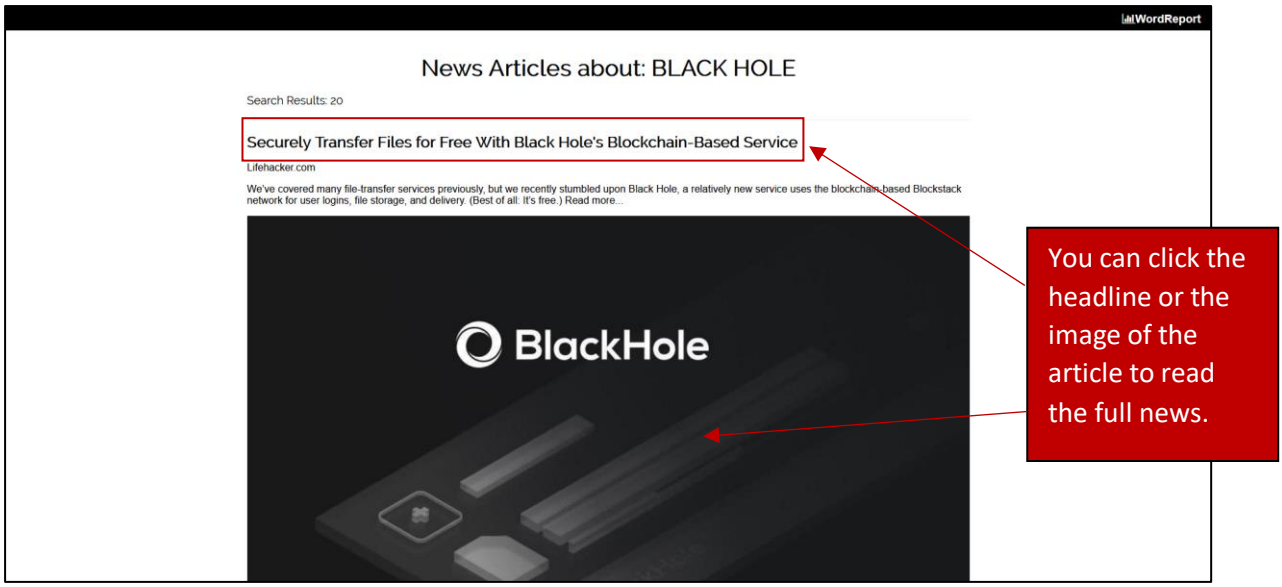
Welcome to WordReport! This web application finds the top 10 news keywords of a country or category that have been frequently mentioned a lot. On the main page there is a choice of choosing categories or searching up a country:

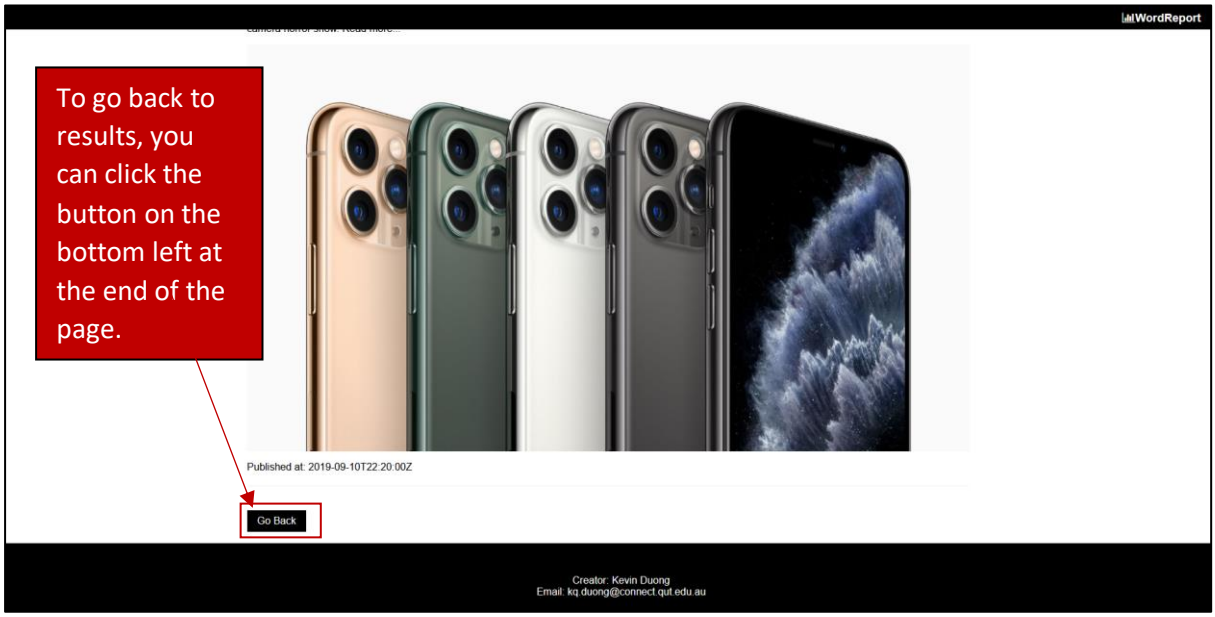
Results Page

Once the page has loaded in with keywords, you can decide whether you want to search again with a different country or go back to the main page and pick a different category. For the main functionality of this page, you can click any keyword that sounds interesting to you.

News Articles

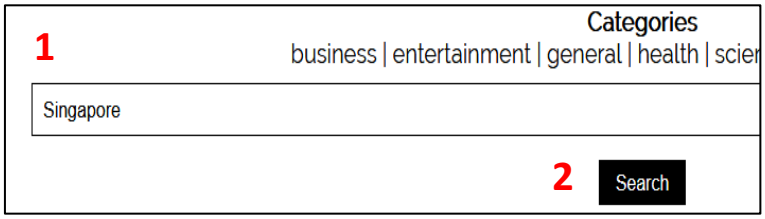
After you decided to click on a term, you will be directed to a news page full of articles related to the word. Feel free to scroll through the webpage to find an article you'd like to read. The maximum limit of articles is 20.

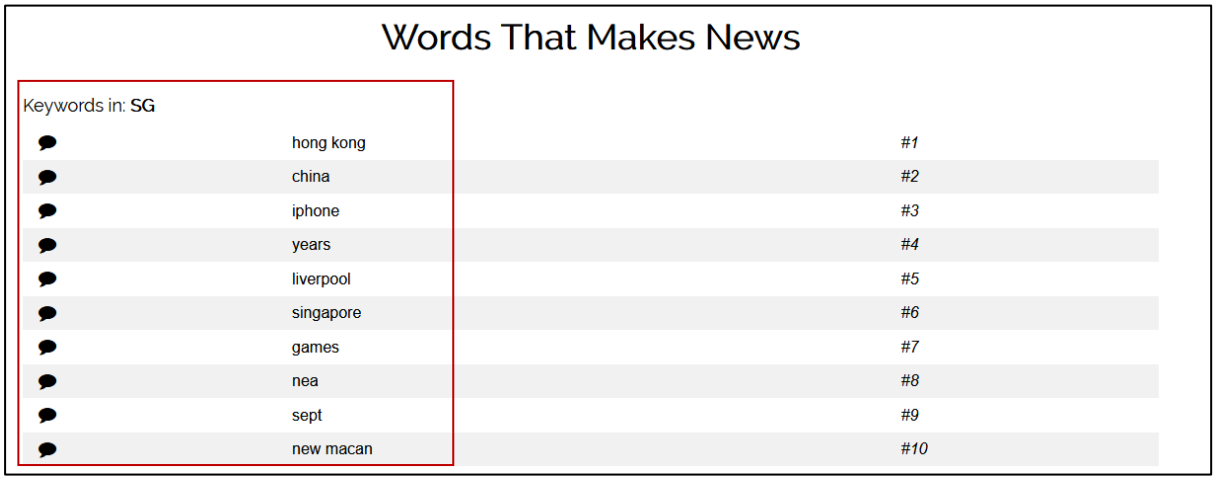




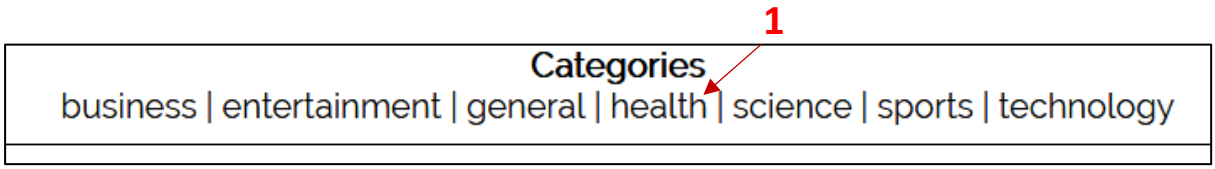
Appendix B – Manual Documentation

01 – Search for words (country)

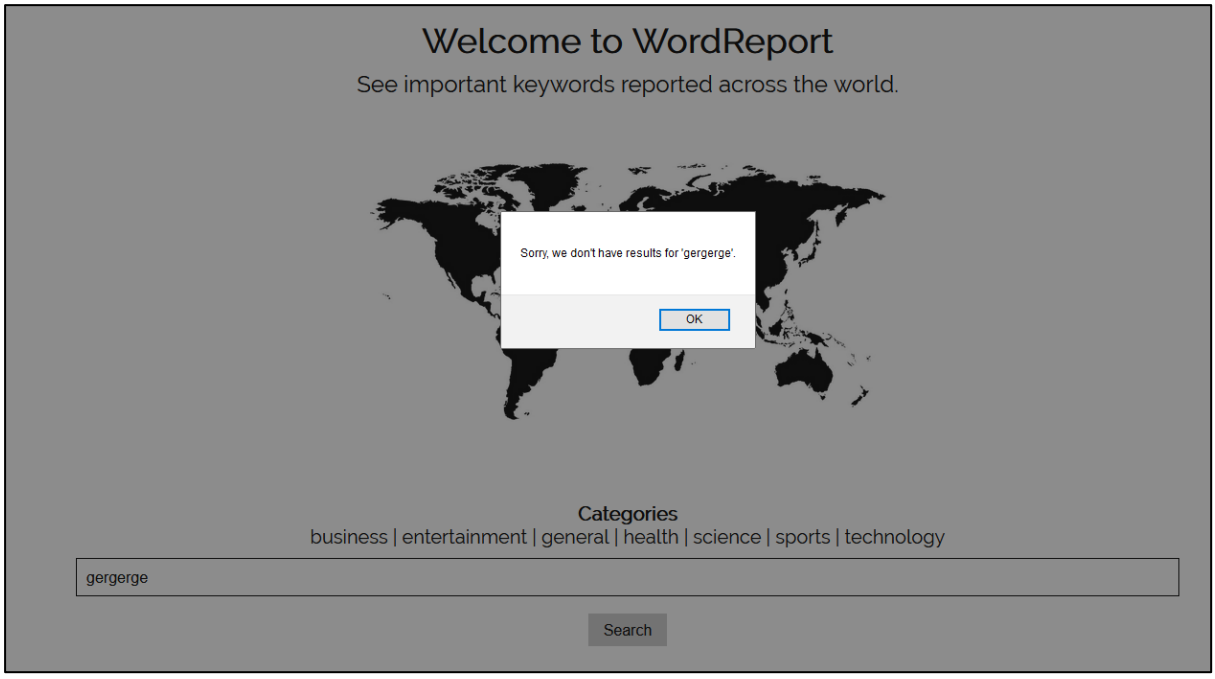




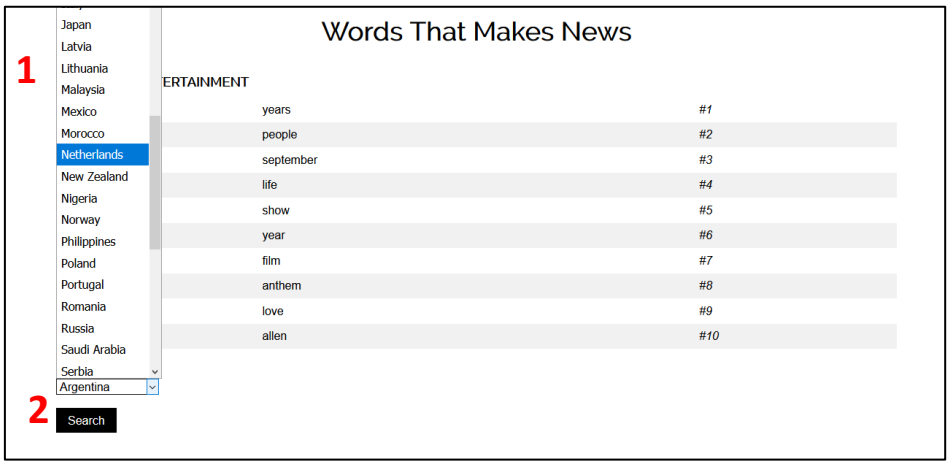
02 – Search for words (category)

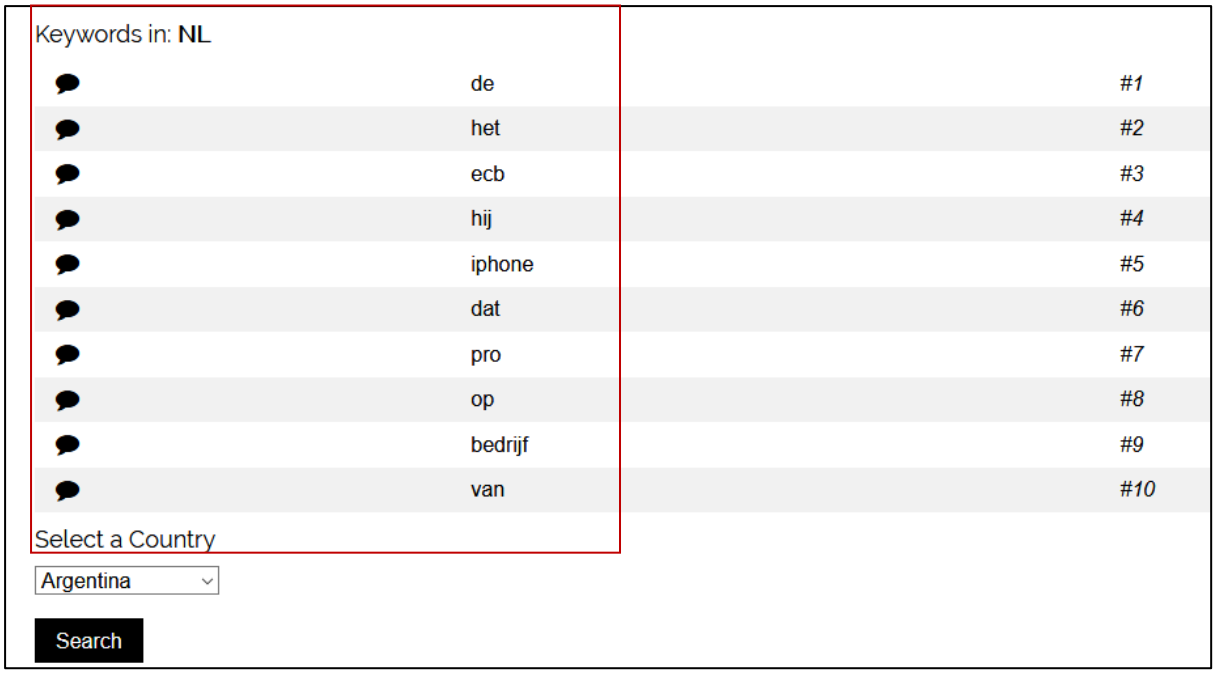


03 – Enter incorrect search

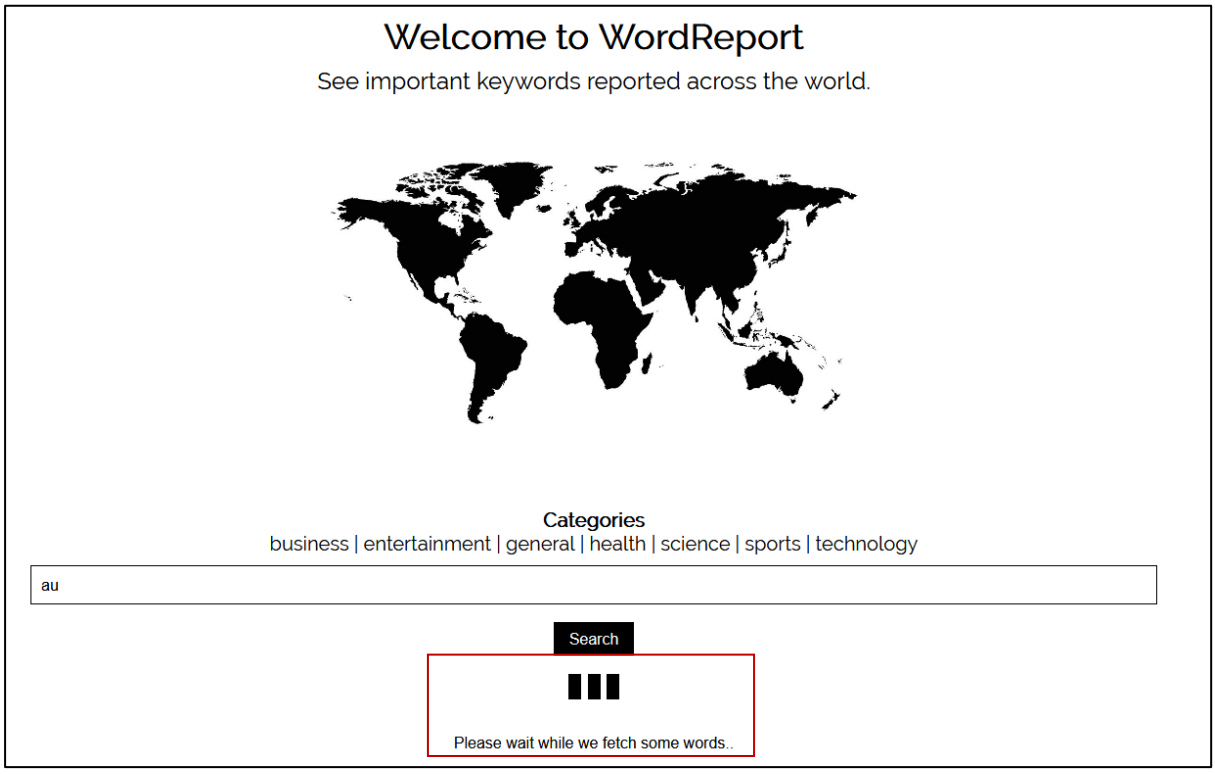


04 – Use dropdown list to search

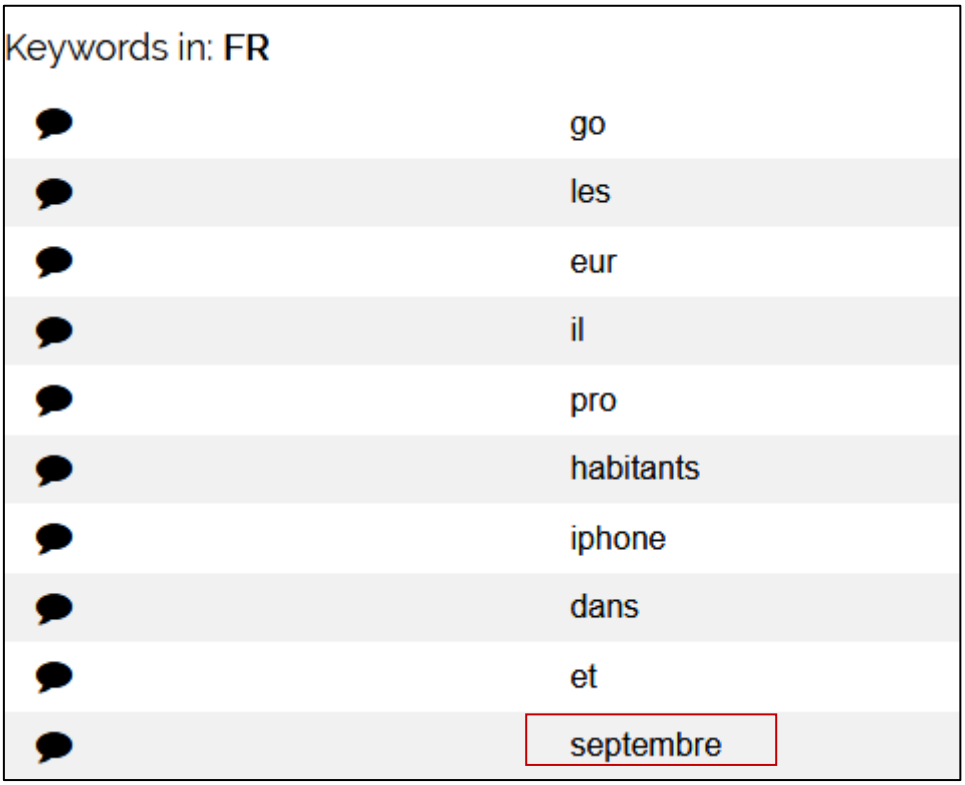




05 – Loading interaction



06 – Click keyword for news



07 – Click article to read full news

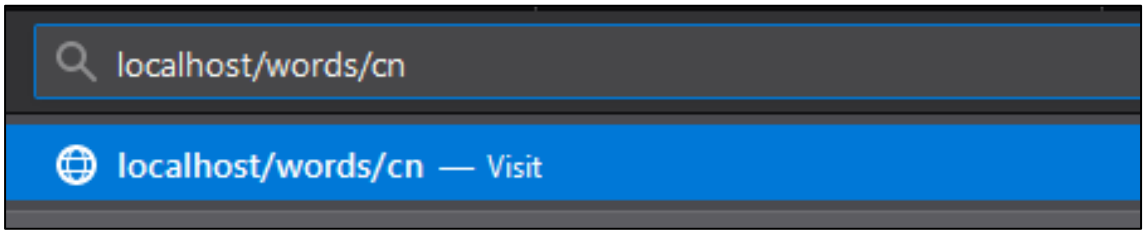
La gazette du mercato : Wissam Ben Yedder sur le Rocher, Adil Rami viré, et Wesley Sneijder retraité

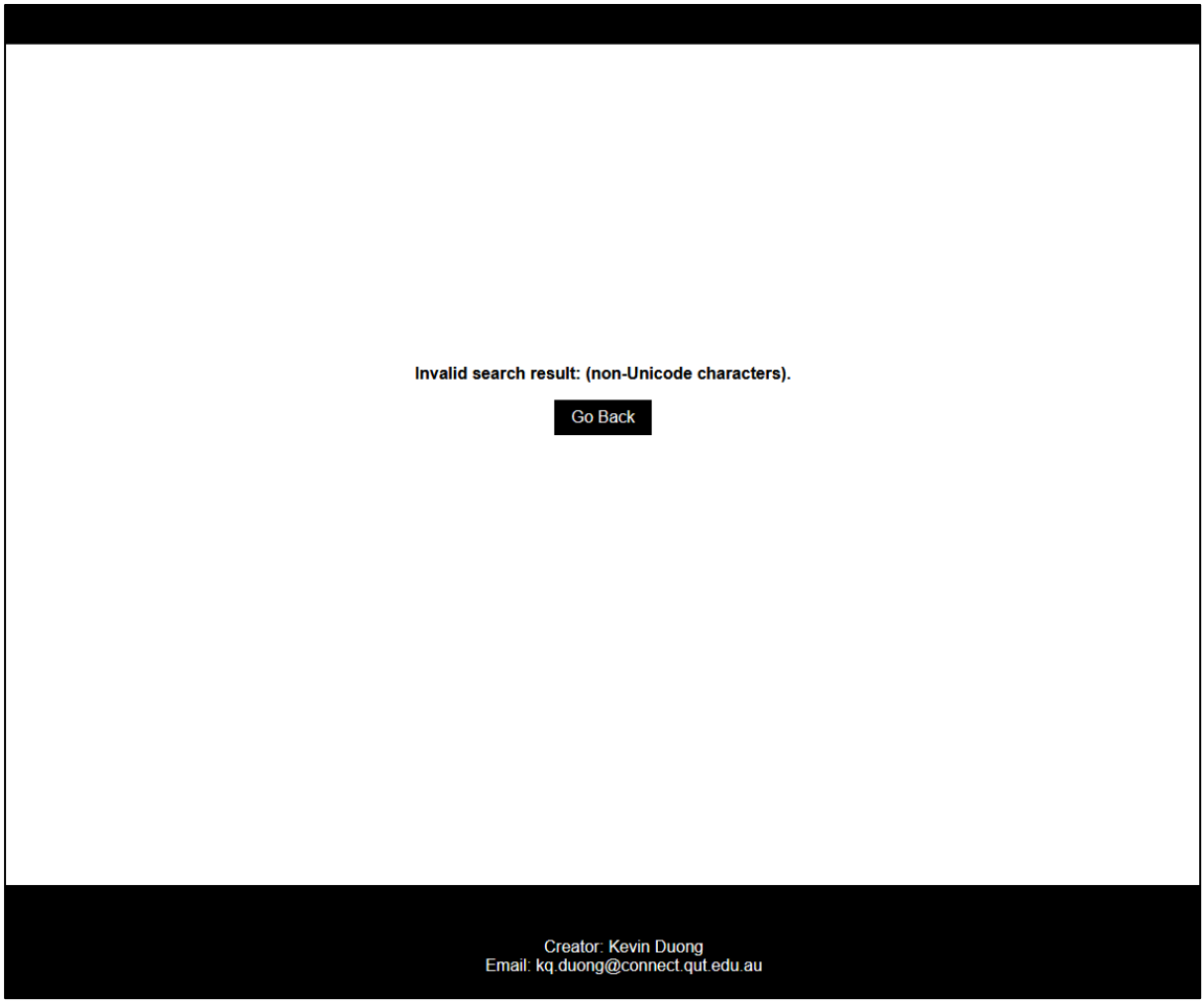
Le Monde

Chaque semaine, « Le Monde » vous fait vivre, à sa manière, le marché estival des transferts, qui dure jusqu'au 2 septembre.

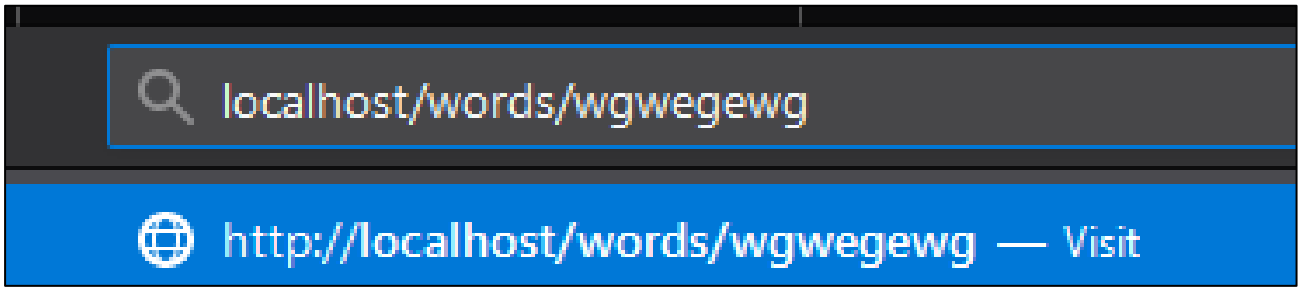
Published at: 2019-08-16T12:57:16Z

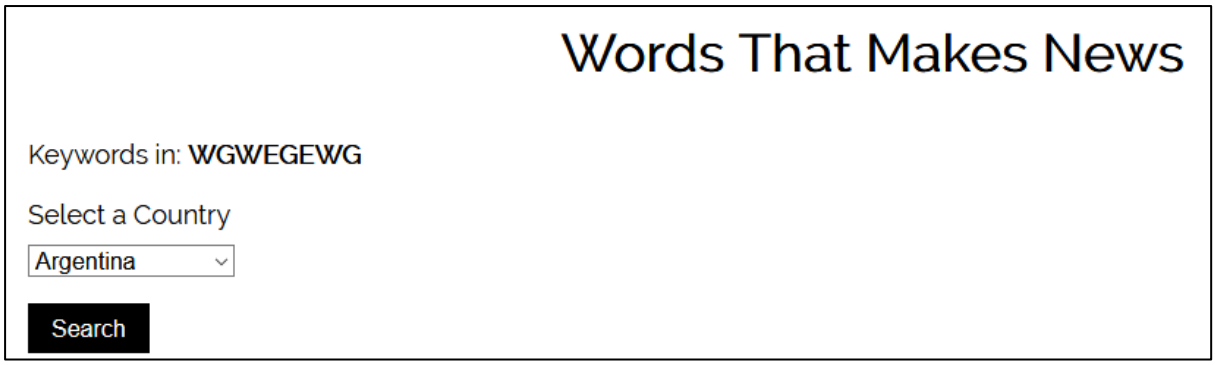
08 – RH: search for invalid country (URL)



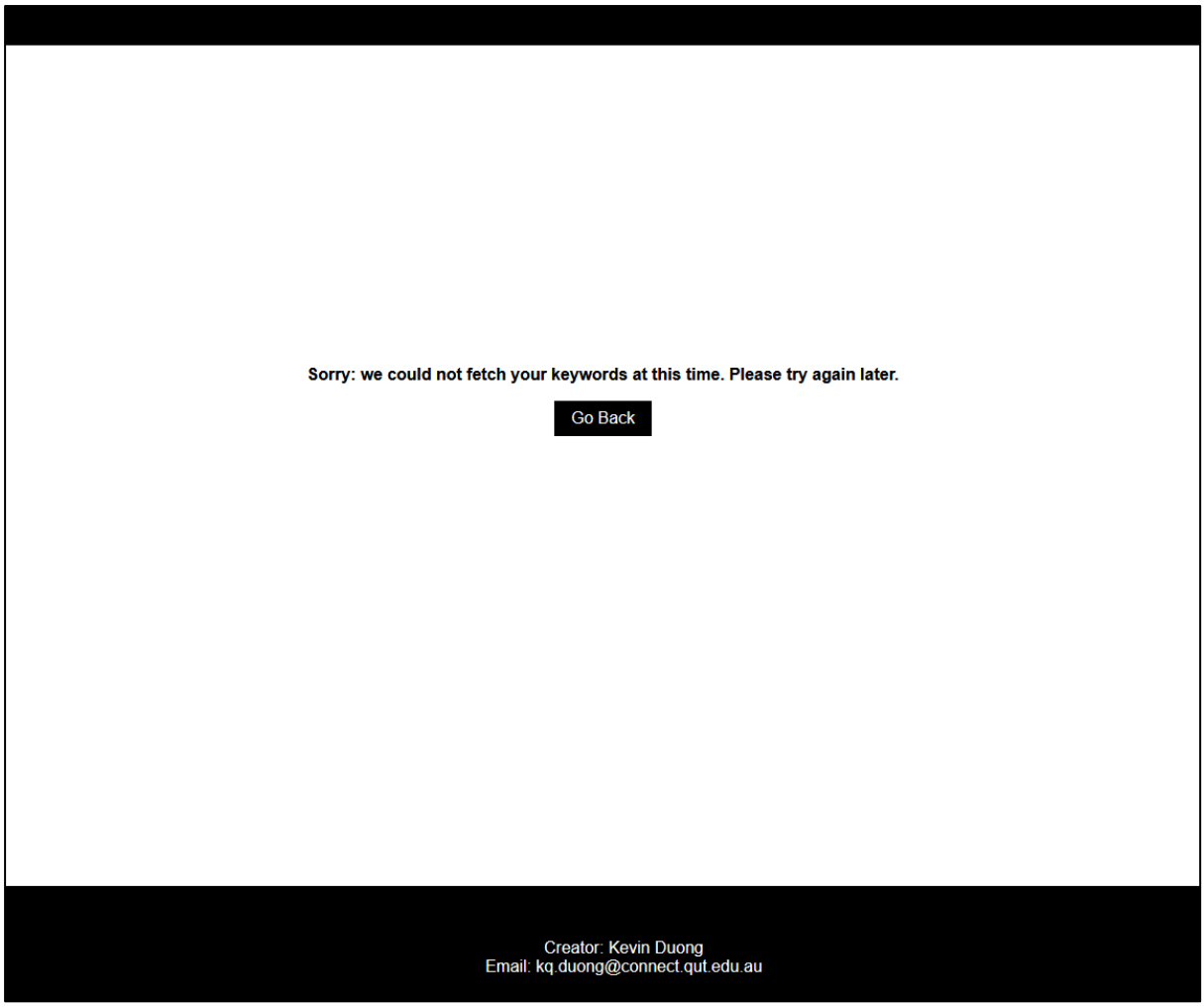


09 – RH: search for incorrect category (URL)





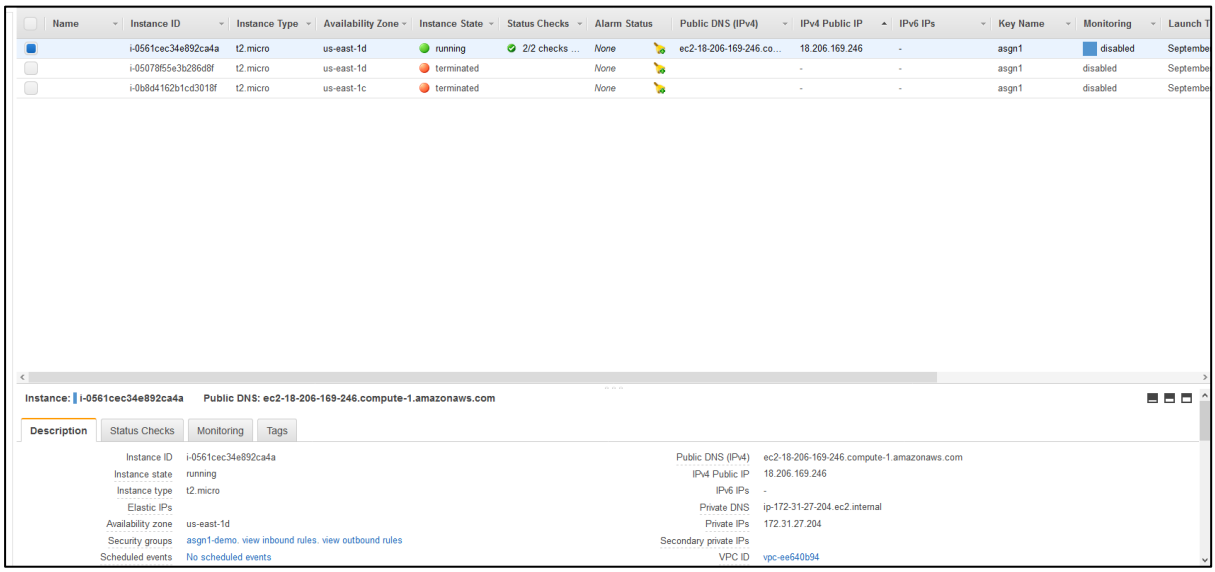
10 – Fetch keywords at a slow connection



Appendix C – Deploying App to Cloud

AWS Public VM Instance

We need to setup a public VM instance first to identify what the host name is for the server – instead using http://127.0.0.1:3000 , we instead use the instance's public IP. An Ubuntu server 18.04 LTS instance from Amazon Web Service will be used to hold our proposed Dockerised app.



The host name from here on will be the IPv4 Public IP + public port:

http://18.206.169.246:3000

Furthermore, changes have been made to connect to the new host throughout the files:

```
8   // Initialise app
9   const app = express();
10
11  const hostname = '54.146.195.172';
12  const port = 80;
```

Building a Docker Image

To build our image, we need to setup a Dockerfile which sets a bunch of commands from us to work with our web application. The following script will be used to build the image:

```
1  FROM node:buster-slim
2  COPY /wordreport .
3  EXPOSE 80
4  RUN npm install
5  CMD ["node", "app.js"]
```

Using Ubuntu virtual machine with Docker installed on the terminal, we are ready to build the image of the app – the image name will be my Dockerhub username + 'asgn1' + a tag.

```
ubuntu@ubuntu-desktop:~/Desktop/asgn1$ docker build -t n9934731/asgn1:1 .
```

The docker will fetch our dependencies and modules to work with the app in the image.

```
ubuntu@ubuntu-desktop:~/Desktop/asgn1$ docker build -t n9934731/asgn1:1 .
Sending build context to Docker daemon  270.8kB
Step 1/5 : FROM node:buster-slim
buster-slim: Pulling from library/node
b8f262c62ec6: Pull complete
cd902351b877: Pull complete
bfd96150ed4c: Pull complete
091f390be83f: Pull complete
38a8448b9186: Pull complete
Digest: sha256:a04ae1bd20545bb1d239a5281026caa21a3af4fc85e50f33467a170ebaf6afcf
Status: Downloaded newer image for node:buster-slim
 ---> 740f5f5b53e3
Step 2/5 : COPY /wordreport .
 ---> 1b118ffc1877
Step 3/5 : EXPOSE 80
 ---> Running in 9802b1a1321e
Removing intermediate container 9802b1a1321e
 ---> 44c4391cb2e2
Step 4/5 : RUN npm install
 ---> Running in cce2d7971d87

> core-js@2.6.9 postinstall /node_modules/core-js
> node scripts/postinstall || echo "ignore"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)

npm WARN cab432-asgn1@1.0.0 No repository field.

added 136 packages from 195 contributors and audited 256 packages in 17.672s
found 0 vulnerabilities

Removing intermediate container cce2d7971d87
 ---> efc76159a27c
Step 5/5 : CMD npm start
 ---> Running in 0684cdbafc36
Removing intermediate container 0684cdbafc36
 ---> c418637e3106
Successfully built c418637e3106
Successfully tagged n9934731/asgn1:1
```

To see if the image is built, we can check docker image directory:

```
ubuntu@ubuntu-desktop:~/Desktop/asgn1$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
n9934731/asgn1      1                   c418637e3106        About a minute ago  178MB
node                buster-slim         740f5f5b53e3        47 hours ago        161MB
```

Pushing Image to Dockerhub

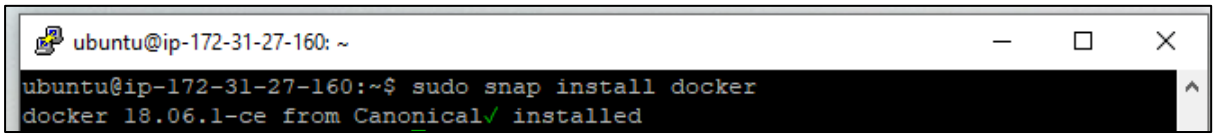
The image can be sent to a user's Dockerhub repository. To do this, we must send a 'push' command to our repository listed in the image directory:

```
ubuntu@ubuntu-desktop:~/Desktop/asgn1$ docker push n9934731/asgn1
The push refers to repository [docker.io/n9934731/asgn1]
466dcf2ed03f: Pushed
1718d62992dc: Pushed
bab9193ada77: Mounted from library/node
227c3f8399ab: Mounted from library/node
ba64ffa52226: Mounted from library/node
627d6bea28f5: Mounted from library/node
2db44bce66cd: Mounted from library/node
1: digest: sha256:3a70489fba596948e2bcafba7caf2b9839142e2f49d5aec4dd8779dfe216606a size: 1787
```

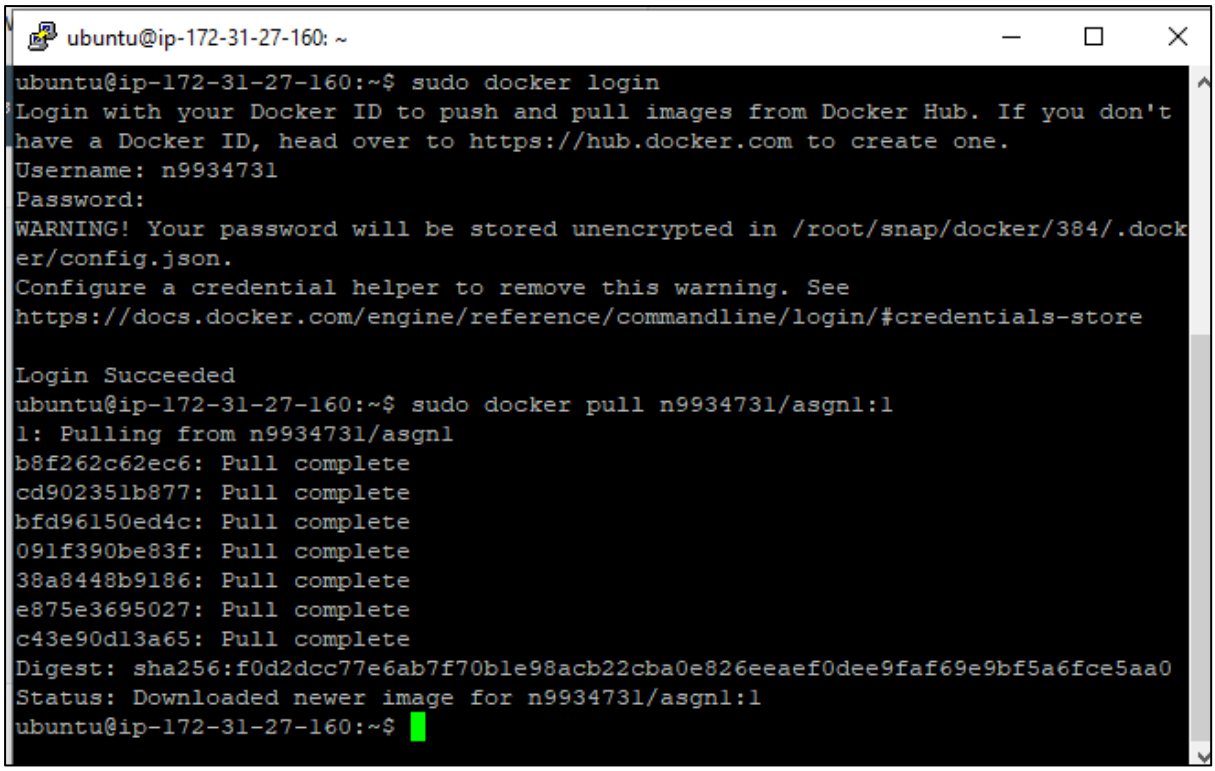
Pulling Image from AWS cloud VM

Given that the public DNS is: `ec2-18-206-169-246.compute-1.amazonaws.com`

We must now connect to our instance that we created in AWS to pull the image. References to how to connect via PUTTY is mentioned in the webpage, but our focus is pulling the image from Dockerhub. First thing is to install Docker in our public VM:

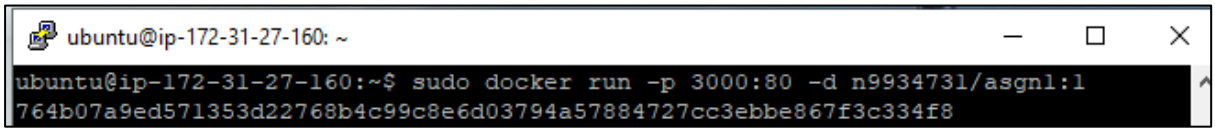


Then login your Dockerhub credentials and pull the image:



Containerising Docker App

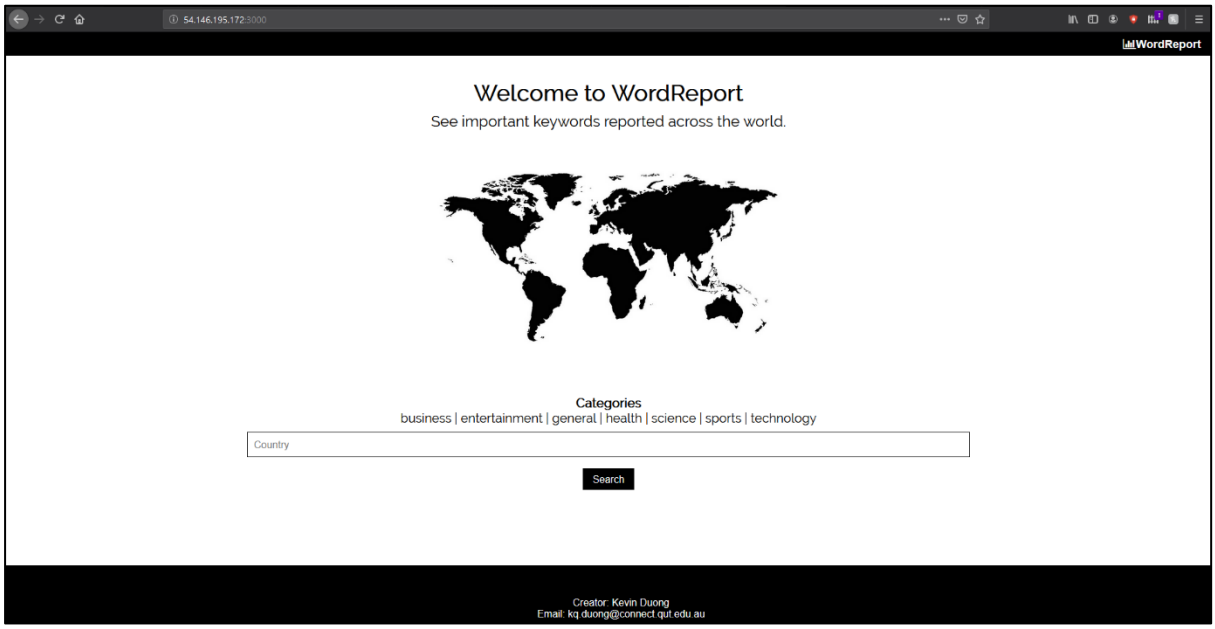
Our final step is to contain our image that we pulled from the repository. We will be running the image at this point and exposing its ports to the public where we can now access the website anywhere, which is now hosted by the public VM.



The virtual machine is now running the app as expected. Port 80 is a private port, but using port 3000, we can access the page here:

http://18.206.169.246:3000

Just like the local version, the website is now running on the virtual machine.



We can also check the container list to see it running:

```
ubuntu@ip-172-31-27-160:~$ sudo docker container list
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS                    NAMES
764b07a9ed57        n9934731/asgn1:1    "docker-entrypoint.s…"   2 minutes ago
     Up 2 minutes        0.0.0.0:3000->80/tcp   lucid_bardeen
```

To end the server, simply stop the container ID:

```
ubuntu@ip-172-31-27-160:~$ sudo docker stop 764b07a9ed57
764b07a9ed57
```