2020

Interwhat

CAB432 Assignment 1

Harry Newton
n1013381-
16/8/2020

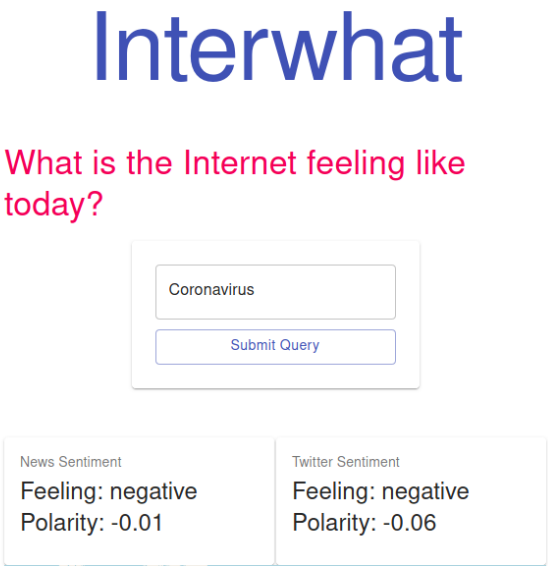
Contents

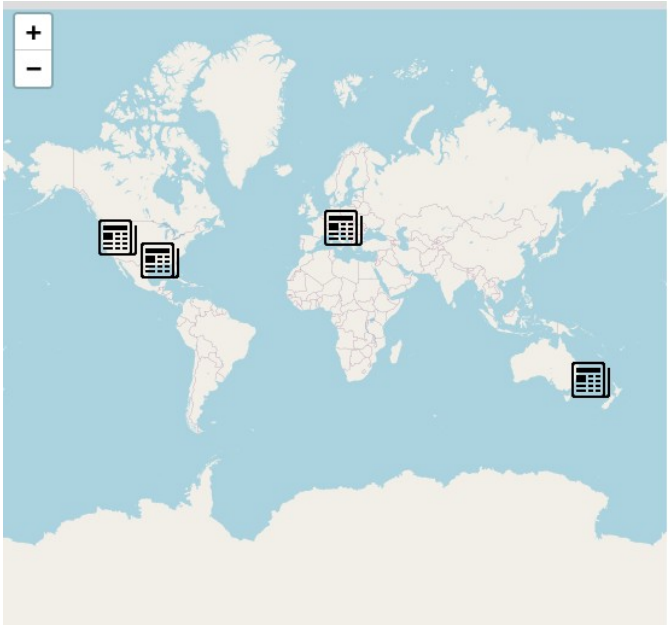
Introduction....1
Mashup Purpose & description....1
Services used....2
Twitter Standard Search API (v.1.1)....2
News API....2
SENTIM-API....2
Leaflet Maps API....2
Mashup Use Cases and Services....3
Topic Search of articles and relevant tweets....3
Visualize Sentiment through location....3
Technical breakdown....4
Architecture and Data Flow....4
Deployment and the Use of Docker....12
Test plan....13
Extensions (Optional)....13
User guide....13
Statement on Assignment Demo....13
Appendices....14

Introduction

Mashup Purpose & description

The purpose of this application is to provide the user an aggregated sentiment analysis popular twitter posts and news articles that relate to a particular topic. Unlike going directly on these news articles and twitter, one can directly get a quantitative measurement of how people are currently feeling about a particular topic. Though, taking into consideration how biased humans can be, it can be best to take a more objective approach to determining the sentiment of these news articles and tweets. Not only does it extract sentiment from a collection of tweets of a particular topic, but it also can display the locations of these sentiments, allowing the user to see how different locations feel about particular topics. The user can view these sentiments through the short description or the google maps component.





Services used

Twitter Standard Search API (v.1.1)

Returns a collection of relevant Tweets matching a specified query – may also be filtered based on popularity or geocoding [and whatever other obvious details we might decide to include]

Endpoint: `https://api.twitter.com/1.1/search/tweets.json`

Docs: https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets

News API

Get breaking news headlines, and search for articles from news sources and blogs all over the web with the news API

Endpoint: `https://newsapi.org/v2/`

Docs: https://newsapi.org/docs

SENTIM-API

Returns a sentiment analysis of text. The sentiment analysis consists an aggregated sentiment, carrying a polarity value and type. The text is split into sentences and are also individually analyzed for sentiment.

Endpoint: `https://sentim-api.herokuapp.com/api/v1/`

Docs: https://sentim-api.herokuapp.com/

Leaflet Maps API

An embedable HTML object which provides an interactive map which can be customised to fit the developers needs.

Endpoint: `https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png`

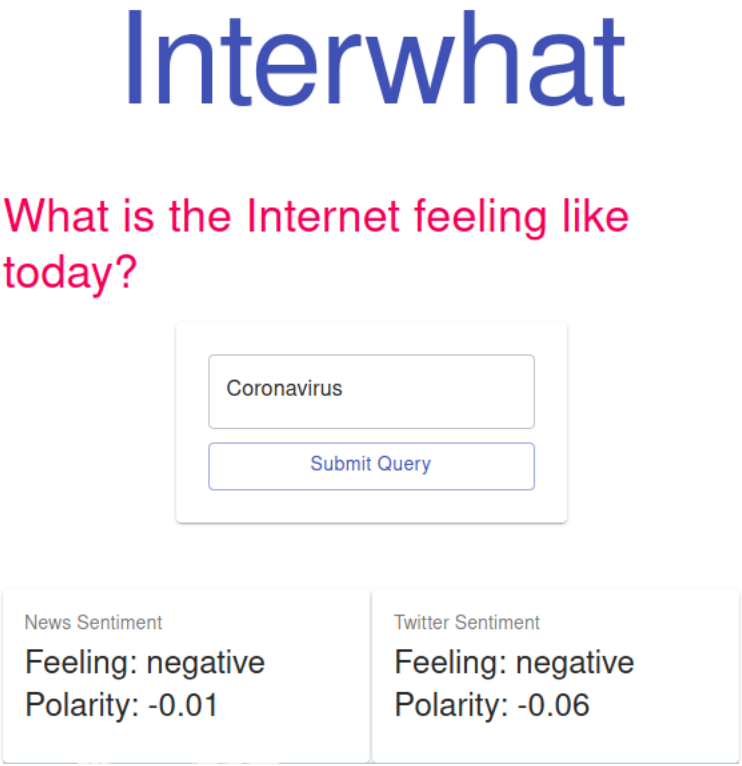
Docs: `https://react-leaflet.js.org/docs/en/intro`

Mashup Use Cases and Services

Topic Search of articles and relevant tweets

As a	Social media enthusiast
I want	To understand the sentiment of social media
So that	I can develop a more aggregated understanding of peoples sentiment towards particular topics

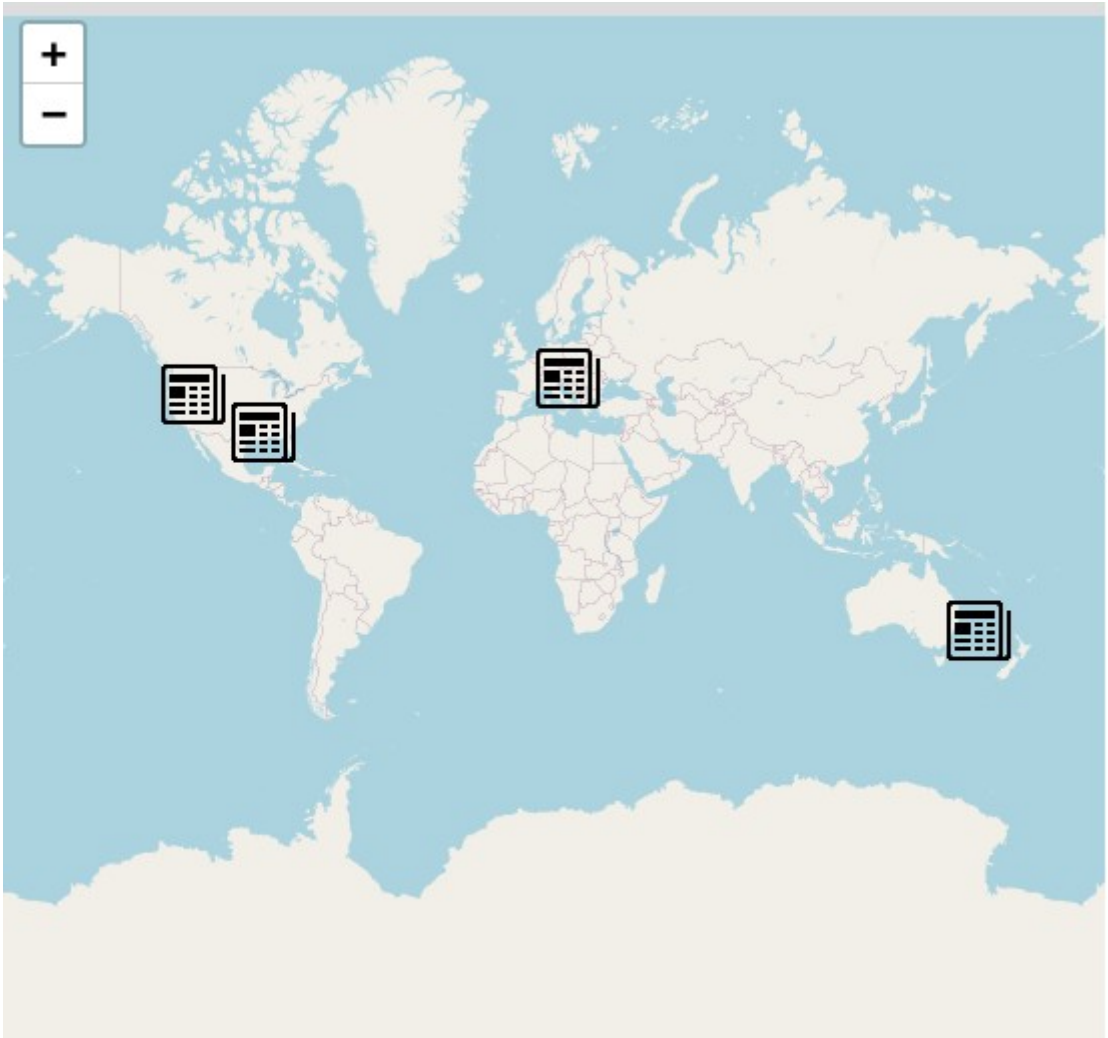
The user will enter any topic/idea/sentence into the query box. Upon submitting, tweets and articles are gathered which are related to this. Aggregated sentiment analysis is performed upon the tweets and articles. This data is then returned back to the client which is displayed in a format representing the aggregated sentiment of twitter and popular news articles.



Visualize Sentiment through location

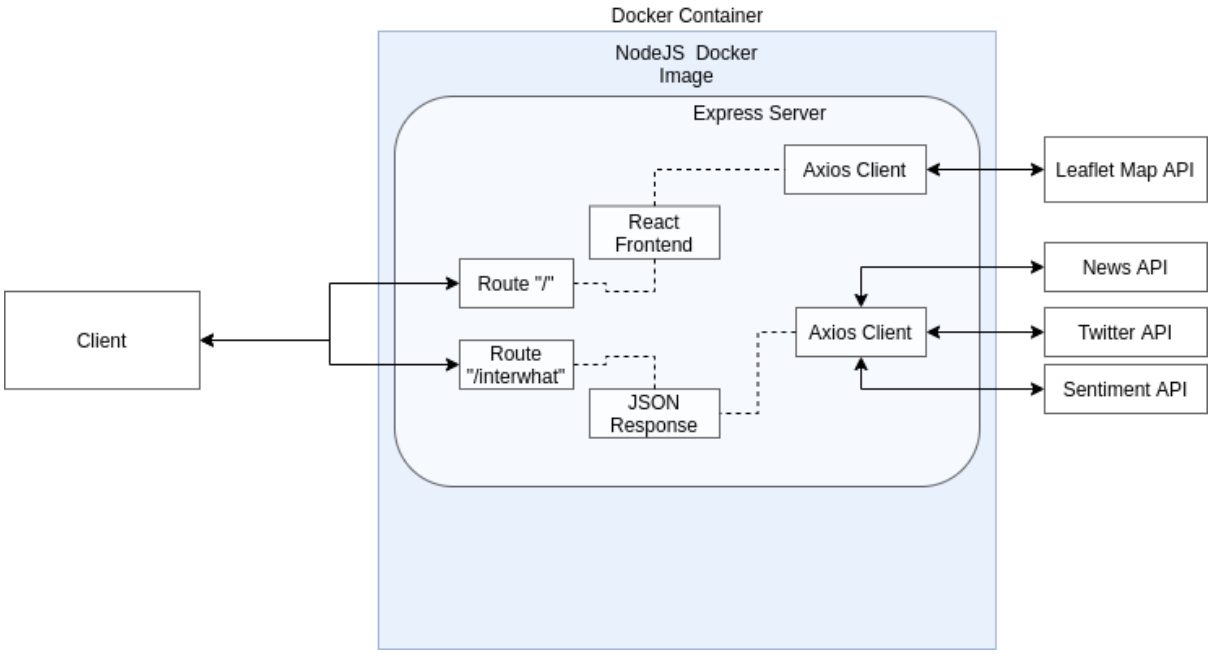
As a	Social media enthusiast
I want	To understand the distribution of sentiment of social media across different location's
So that	I can have an understanding of which particular counties/regions across the world, hold sentiments towards particular topics.

The user will enter any topic/idea/sentence into the query box. Upon submitting, tweets and articles are gathered which are related to this. Sentiment analysis is applied individually to each tweet and article. The location is extracted from the articles and tweets. The individual sentiment and location of each tweet and article is returned to the client and is displayed on a map for the user to interact with.

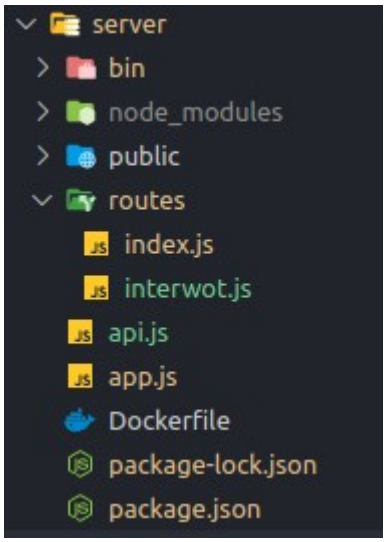


Technical breakdown

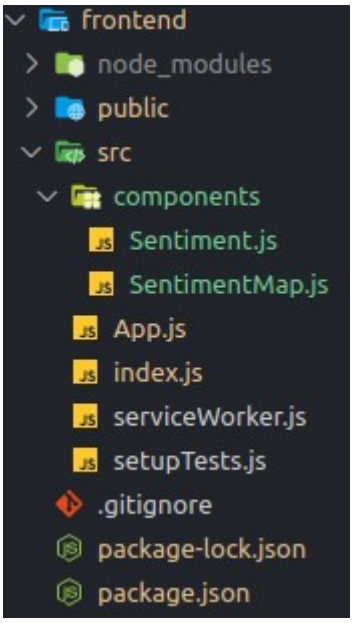
Architecture and Data Flow



Server Code Organization



Frontend Code Organization



Server Code:

```
/* POST which takes a body of text and inserts into the twitter API */
router.post("/interwhat", async (req, res, next) => {
  let query = req.body.text;
  if (query) {
    //TODO: Include word tokenization here

    // News analysis
    let news = api.get_articles(query).then(async (response) => {
      let aggregated = await api.aggregated_article_sentiment(response);
      let location = await api.news_sentiment_location(response);
      return {
        aggregated,
        location,
      };
    });

    // Twitter analysis
    let tweets = api.twitter_standard_search(query).then(async (response) => {
      let aggregated = await api.aggregated_twitter_sentiment(response);
      let location = await api.twitter_sentiment_location(response);
      return {
        aggregated,
        location,
      };
    });

    // Wait for the twitter and news functions to finish
    Promise.all([news, tweets]).then((data) => {
      if (data[0] || data[1]) {
        res.status(200).json({ news: data[0], tweets: data[1] });
      } else {
        res.status(404).json({ error: true, message: "No data found" });
      }
    });
  } else {
    res.status(400).json({
      error: true,
      message: "Request body invalid - text required",
    });
  }
});
```

The server consists of two endpoints, the index route "/" and the api route "/interwhat". The react application is served on the "/" route and the code above shows the code behind the mechanics of the "/interwhat" route. It creates two promises which retrieve the news and twitter API data. Upon completion of both of the promises, if either of the data is successful, the information is sent to the client. If the data retrieval is not successful or the user has sent an empty query, an error is raised and the client is notified.

```javascript
// TWITTER API FUNCTIONS
// Requests the first 100 most popular twitter statuses on a particular general search
const twitter_standard_search = async (query) => {
  return axios({
    method: "get",
    url: `${TWITTER_API}/search/tweets.json?q=${query}&count=100`,
    headers: {
      Authorization: `Bearer ${TWITTER_BEARER_TOKEN}`,
      Cookie:
        'personalization_id="v1_YJ+TPi2r5WXMzfrGGX7WlA=="; guest_id=v1%3A159824405664029578',
    },
  })
    .then((response) => response.data.statuses)
    .catch((err) => console.error(err));
};

// SENTIMENT ANALYSIS API FUNCTIONS

// Sends text data to the sentiment API to be analysed
const sentiment_analysis = async (text) => {
  return axios({
    method: "post",
    url: "https://sentim-api.herokuapp.com/api/v1/",
    headers: {
      "Content-Type": "application/json",
    },
    data: JSON.stringify({
      text: text,
    }),
  })
    .then((sentiment) => sentiment.data.result)
    .catch((err) => console.error(err));
};
```

This code is within the "api.js" file and the two functions being shown retrieve the data from the news and twitter API.

```
// Joins all of the text from all of the statuses and processes it through the sentiment API
const aggregated_twitter_sentiment = async (statuses) => {
  return await sentiment_analysis(
    statuses.map((status) => status.text).join("\n")
  );
};

// Applies sentiment analysis to each status and pairs it with its relevant location
const twitter_sentiment_location = async (statuses) => {
  return Promise.all(
    statuses.map(async (status) => {
      if (status.coordinates) {
        let sentiment = await sentiment_analysis(status.text);
        return {
          sentiment,
          coordinates: status.coordinates,
        };
      }
    })
  );
};

// NEWS API
const get_articles = async (title) => {
  return axios({
    method: "get",
    url: `https://newsapi.org/v2/top-headlines?q=${title}&apiKey=${NEWS_API_KEY}`,
  })
    .then((response) => response.data.articles)
    .catch((err) => console.error(err));
};

const aggregated_article_sentiment = async (articles) => {
  return await sentiment_analysis(
    articles.map((article) => article.description).join("\n")
  );
};
```

This code utilizes the two API functions and collects an “mashes” the two APIs together. These extracts sentiment and location data from the news and twitter sources.

Frontend Code:

```
// State for holding topic user entered
const [topic, setTopic] = useState("");
// Default state for sentiment data
const [sentimentData, setSentimentData] = useState({
  news: { aggregated: {}, location: [] },
  twitter: { aggregated: {}, location: [] },
});
// Axios hook to manage requests better
const [{ data, loading, error, response }, execute] = useAxios(
  {
    method: "post",
    url: "http://localhost:8000/interwhat",
    // url: "http://54.153.162.105//interwhat",
    headers: {
      "Content-Type": "application/json",
    },
  },      You, 10 minutes ago • Uncommitted changes
  { manual: true, useCache: false }
);
```

The code above shows the hooks which are related to data collection. An axios hook was utilized to ease the process of interacting with the backend of the server. Default data is set in order for components relying on the data to resolve "undefined" or "null" data.

```
// Function for form
const onSubmit = async (text) => {
  // Merges this object with the existing one in the hook and sends the data to the API
  execute({
    data: JSON.stringify(text),
  });

  setTopic(text.text);
};
useEffect(() => {
  console.log(sentimentData.news);
});

// Update the sentimentData state variable upon data response from API
useEffect(() => {
  setSentimentData({ ...data });
}, [data]);
```

The code above handles the user query submission. Upon submission, the query is sent to the server. The useEffect functions wait for the response data to be updated and in turn, update the sentimentData hook.

```
{loading ? (
  <PacmanLoader
    size={25}
    style={{ margin: "2px" }}
    color={"#123abc"}
  />
) : (
  <Sentiment data={sentimentData} topic={topic} error={error} />
)}
```

This code handles displaying the loading component and the Sentiment component . This is done using one of the states given by the axios hook, allowing react to know when the data is still being retrieved.

```
<Grid item>
  {news && news.aggregated ? (
    <Card className={classes.root}>
      <CardContent>
        <Typography
          className={classes.title}
          color="textSecondary"
          gutterBottom
        >
          News Sentiment
        </Typography>
        <Typography variant="h5" component="h3">
          Feeling: {news.aggregated.type}
        </Typography>
        <Typography variant="h5" component="h3">
          Polarity: {news.aggregated.polarity}
        </Typography>
      </CardContent>
    </Card>
  ) : (
    <Card className={classes.root}>
      <CardContent>
        <Typography
          className={classes.title}
          color="textSecondary"
          gutterBottom
        >
          News Sentiment
        </Typography>
        <Typography variant="h5" component="h3">
          No news sentiment
        </Typography>
      </CardContent>
    </Card>
  )}
```

The code snippets below contain code from "Sentiment.js". The image on the left shows how the aggregated sentiment value from the news and twitter API's are displayed. The image on the right shows how the sentiment locations are displayed, which are through the "SentimentMap.js" Component. The last image on the bottoms shows the code regarding the map. It generates markers for each tweet and article, with its associated sentiment.

```
        <SentimentMap
          newsLocations={news && news.location}
          twitterLocation={twitter && twitter.location}
        />
      </React.Fragment>
    );
  }

  // When no data is available
  return <Empty description={<span>No Sentiment Data</span>} />;
```

```
function SentimentMap({ newsLocations, twitterLocations }) {
  return (
    <Map center={[0, 0]} zoom={2} style={{ height: "100vh", width: "100vh" }}>
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
      />
      {generateMarkers(
        newsLocations || [],
        <BsNewspaper size={32} style={{ fill: "black" }} />
      )}
      {generateMarkers(
        twitterLocations || [],
        <AiOutlineTwitter size={32} style={{ fill: "blue" }} />
      )}
    </Map>
  );
}
```

Deployment and the Use of Docker

In regards to the docker file, the image being used was **node:dubnium**, which includes node by default. The exposed port will be 80, serving the application as a standard website. The application will be deployed on an AWS EC2 Ubuntu instance. An image of the dockerfile can be found in the appendix.

Test plan

Task	Expected Outcome	Result	Screenshots
Search for a query	Aggregated sentiment is displayed for news and twitter	PASS	2
Search for a query	Location sentiment is displayed for news and twitter	PASS	3
Search for a query with empty input	Form will notify user text input is required	PASS	4
Enter a query with nonsensical input	No data will be returned, as displayed with empty component	PASS	5

Difficulties / Exclusions / unresolved & persistent errors /

One of the major difficulties was working with the Twitter API. This is because one of the use cases of the app is to visualize the sentiment of tweets and its corresponding locations. Unfortunately, or lets say rationally, people aren't as willing to share their location at all times with Twitter. This prevents the app from having a practical use case.

One of the other problems was the handling of errors with deeply nested JSON. As expected javascript isn't good at handling undefined variables and so it becomes increasingly more complex to handle nested JSON data from the server.

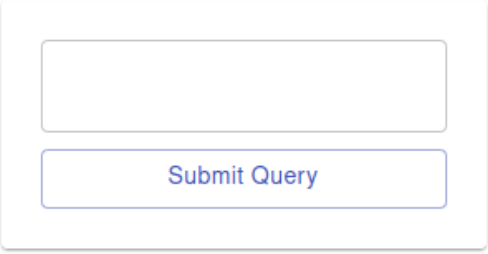
Conditional rendering was another vice that has not developed well with the react frontend. As mentioned before, handling deeply nested JSON with conditional rendering becomes difficult.

In regards to the aesthetic of the frontend, there are some odd bugs that make some components look strange upon load. For instance, the Material-UI button has a weird semi-pressed state once the application is loaded, which can be unappeasing to some users.

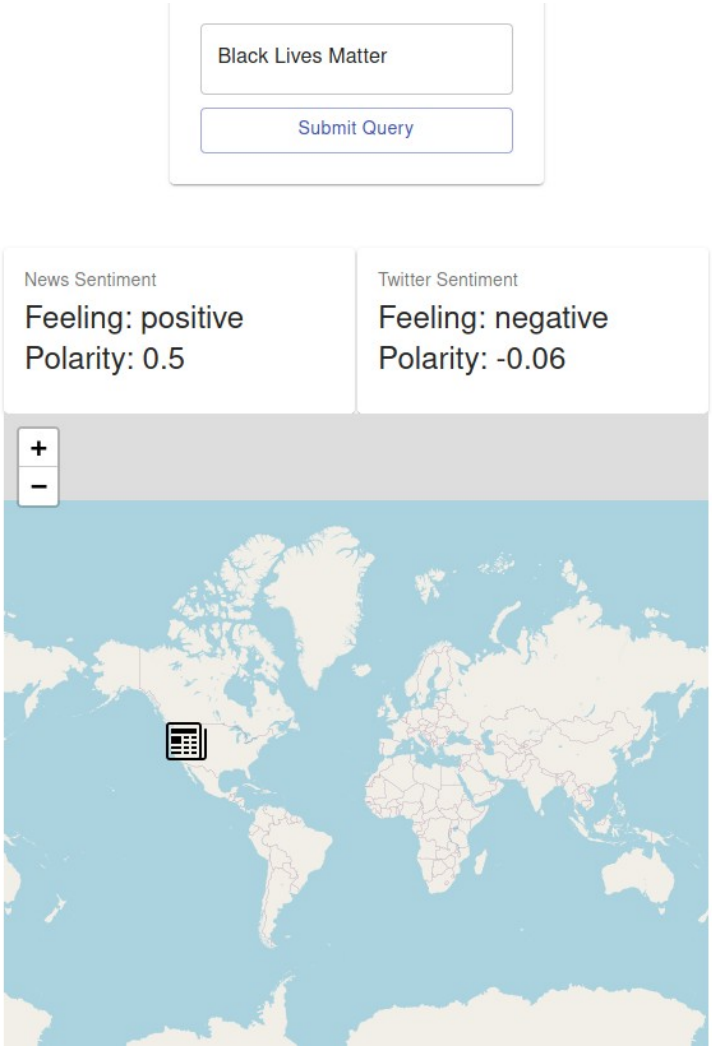
User guide

Interwhat

What is the Internet feeling like today?



When the user first opens the website, they are presented with a 1 page application. They can enter a query into the text box near the top of the page.



Upon submitting a query, for example a controversial political topic in America right now “Black Lives Matter”, it displays the associated sentiment of news and twitter sources, alongside the locations of the news and twitter sources.

Statement on Assignment Demo

Here is a link to my video which has been uploaded online.

https://drive.google.com/file/d/1fzZhGpmEe_Fpsu8Io68ZcKEEqO2QnUr3/view?usp=sharing

References

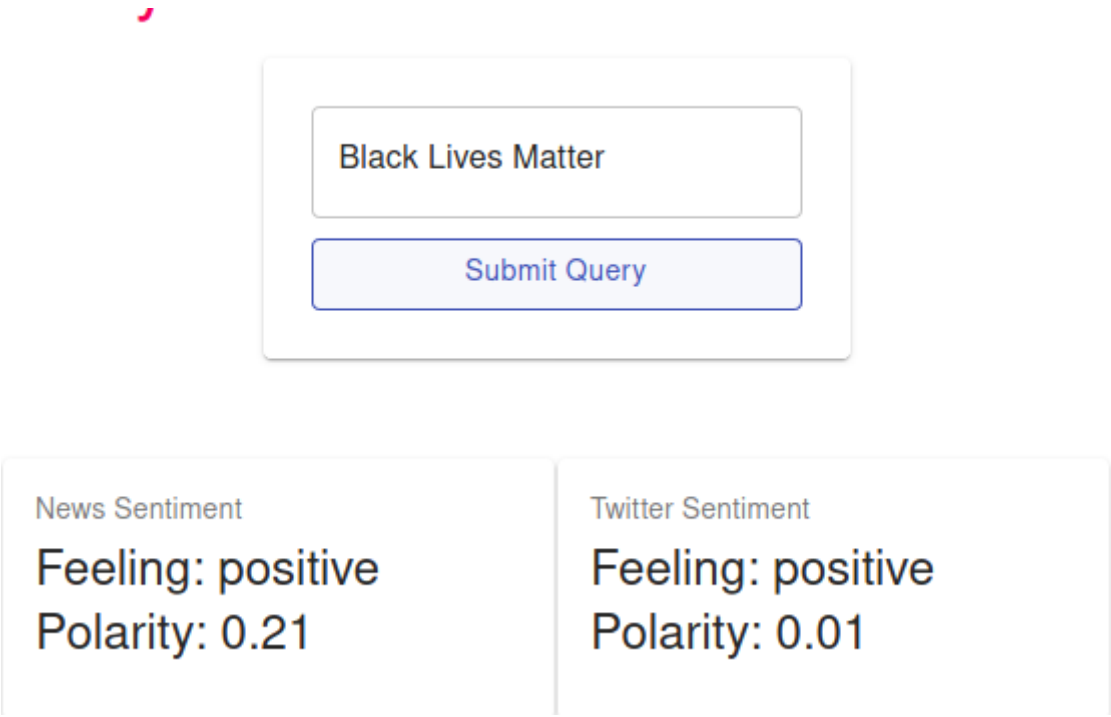
Use a standard approach to referencing – see the guidance at https://www.citewrite.qut.edu.au/cite/.

Appendices

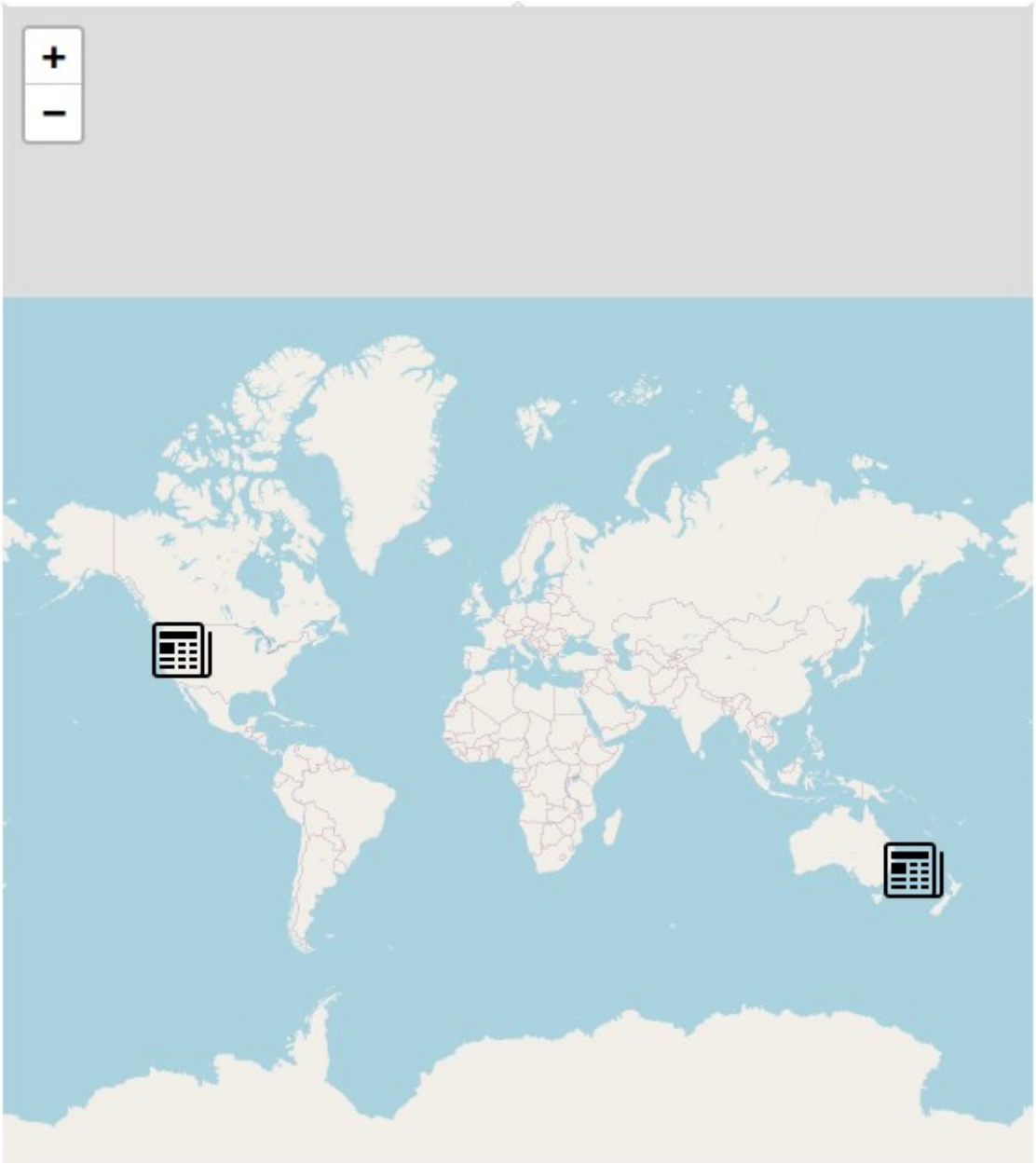
1. Dockerfile

```
You, 21 days ago | 1 author (You)
FROM node:dubnium

# Copy app source
COPY . /src

# Set the work directory to /src
WORKDIR /src

# Install the dependencies
RUN npm install

# Set environment variable port to 80
ENV PORT=80

# Expose the applicatoin to the outside world
EXPOSE 80
      You, 21 days ago • first commit
# Start command as per package.json
CMD ["npm", "start"]
```

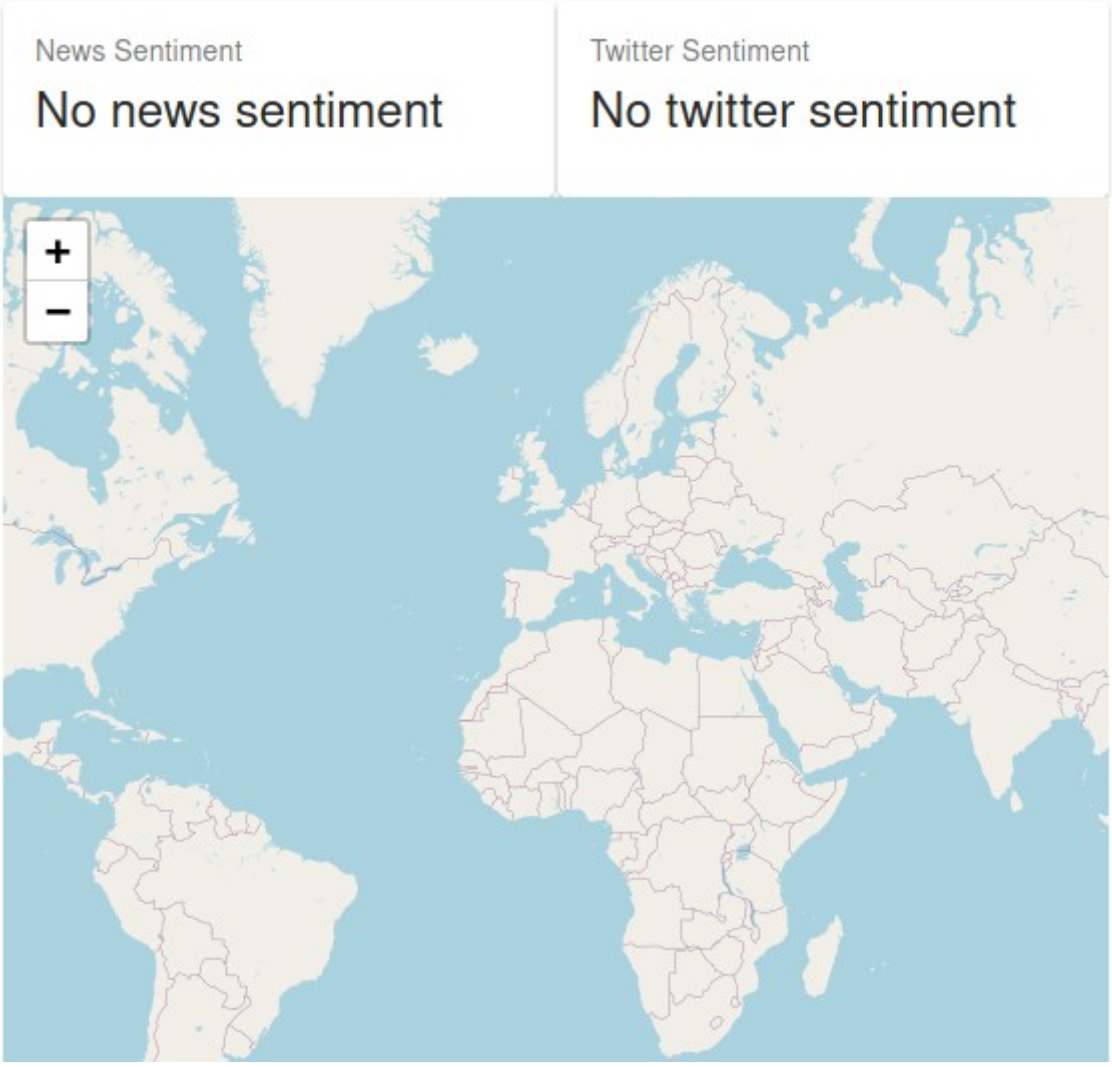
2.


Black Lives Matter
Submit Query
News Sentiment
Feeling: positive
Polarity: 0.21
Twitter Sentiment
Feeling: positive
Polarity: 0.01

3.



4.


News Sentiment
No news sentiment
Twitter Sentiment
No twitter sentiment
+
−