






Universitetet  
i Stavanger

Faculty of Science and Technology

## BACHELOR'S THESIS

Study program/Specialization:  Bachelor's in Computer Science /	Spring semester 2021  Open / Restricted access
Writer(s): Awalle Mohamed, Jone Aabø Lorentzen, Ramtin Hafezinejad	
Faculty supervisor:  Vinay Jayarama Setty	  Jone Lorentzen 
Thesis title: Coverage of fake news  Engelsk tittel: Coverage of fake news	
Credits(ECTS): 20	
Key words:  Web application for coverage of fake news on social media	Pages: 59  + vedlegg/annet:  Stavanger 15. mai 2021



*“Beware of false knowledge, it is more dangerous than ignorance”*

— George Bernard Shaw

## *Abstract*

In recent years, we have seen a significant increase in the spread of misinformation, also known as fake news, especially on social media. Any user could share a claim or a misrepresentation of something, and in a few hours, it would have gained a lot of traction over the whole world. This does not mean that every piece of misinformation will be spread across the world, but with the current technology, everyone has the opportunity. The growing problem with fake news is not going away anytime soon, but with this thesis we want to create a web application representing the spread of fake news across different social media.

## *Acknowledgements*

We would like to thank our supervisor Vinay Jayarama Setty at the Department of Electrical Engineering and Computer Science at University of Stavanger. Selecting a thesis seemed scary at the first glance, but we knew this thesis was something we all had an interest for. With Vinay's expertise within programming and fake news we have learned a lot throughout the project.



# Contents

<b>Abstract</b>	<b>iv</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Abbreviations</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation	1
1.2 Problem Description	2
1.3 Goal	2
1.4 Use cases for the application	3
1.5 Related Work	3
1.6 Deployment	4
1.7 Contributions	4
1.8 Outline	5
<b>2 Background</b>	<b>7</b>
2.1 Fake News	7
2.2 Programming Languages	8
2.2.1 Python	8
2.2.2 HTML	8
2.2.3 CSS	8
2.2.4 Bootstrap	9
2.2.5 JavaScript	9
2.2.6 Flask	9
2.3 Vue	9
2.3.1 Vue.js	9
2.3.2 Vuex	10
2.3.3 Chartkick.js	10
2.3.4 D3.js	10
2.4 JSON	11
2.5 Twitter	11
2.6 Reddit	11

<b>3</b>	<b>System Architecture</b>	<b>13</b>
3.1	Introduction . . . . .	13
3.2	Design Pattern . . . . .	14
3.3	Using the API's for retrieval of data . . . . .	14
3.3.1	Twitter API . . . . .	14
3.3.2	Reddit API . . . . .	16
3.4	Back-end . . . . .	18
3.4.1	Flask Back-end . . . . .	18
3.4.2	Pagination . . . . .	19
3.4.3	Error Catching . . . . .	21
3.5	Formatting the data . . . . .	21
3.5.1	Formatting Twitter data . . . . .	22
3.5.2	Formatting Reddit data . . . . .	23
3.6	Communication between back-end and front-end . . . . .	23
3.7	Sequence Diagram . . . . .	26
<b>4</b>	<b>Front-End Design</b>	<b>27</b>
4.1	Introduction . . . . .	27
4.2	Pages and design . . . . .	27
4.2.1	Landing Page . . . . .	28
4.2.2	Your trackers . . . . .	29
4.2.3	Dashboard . . . . .	29
4.3	Components . . . . .	31
4.3.1	Tooltip for Components . . . . .	32
4.3.2	Search list . . . . .	32
4.3.3	Engagement . . . . .	33
4.3.4	Line chart . . . . .	34
4.3.5	Bar chart . . . . .	35
4.3.6	Geo chart . . . . .	35
4.3.7	Node network . . . . .	36
4.3.8	Top Posts and Most Influential Users . . . . .	37
4.3.9	Reddit Word Cloud . . . . .	38
4.3.10	Twitter Sentiment Component . . . . .	38
4.3.11	Accessing the data in the components . . . . .	40
<b>5</b>	<b>Discussion</b>	<b>43</b>
5.1	Problems and challenges . . . . .	43
5.2	Experimental Evaluation . . . . .	44
5.2.1	Sentiment Analysis . . . . .	44
5.2.2	Performance Evaluation . . . . .	47
5.2.3	Feedback Survey . . . . .	49
5.3	Further Development . . . . .	50
5.3.1	Premium Twitter API . . . . .	50
5.4	Conclusion . . . . .	51



**List of Tables**

**55**

**Bibliography**

**59**



# Abbreviations

<b>Acronym</b>	<b>What (it) Stands For</b>
<b>UiS</b>	<b>University of Stavanger</b>
<b>API</b>	<b>Application Programming Interface</b>
<b>JSON</b>	<b>JavaScript Object Notation</b>
<b>JS</b>	<b>Java Script</b>
<b>CSS</b>	<b>Cascading Style Sheets</b>
<b>HTTP</b>	<b>Hypertext Transfer Protocol</b>
<b>NLTK</b>	<b>Natural Language Toolkit</b>
<b>UI</b>	<b>User Interface</b>



# Chapter 1

## Introduction

### 1.1 Motivation

In this day and age of social media and constant information flooding, it is not always easy to detect fake news. The primary source of news for many people today, especially the young, is social media. Social media does not only work as a tool for interacting with people worldwide but also as a tool to see what is going on in the rest of the world. The fact that young people are the most exposed to social media today, combined with the fact that many may not be as critical of what they read on the internet as the older generations, is problematic.

Domestically in Norway, one of the most known stories of fake news and its effects would be the school election at the local high school in Lillestrøm. A TV show on the state channel NRK, called «Folkeopplysningen» used a stream of fake news on social media to convince the students to vote for the party that had been the smallest one in the previous elections. The experiment was, of course, revealed to the students five days before the actual election, and thus, the smallest party did not win; nevertheless. When interviewed by the state channel, the students did say that the manipulation of the election gave them a more positive sight of the smallest party.

Globally there have been speculations about fake news being used in several big elections; the 2016 election in the US where Trump won, Brexit, etc. Although there have been several fake news stories regarding Trump in the 2016 election where some may speculate that this was part of him winning, we cannot say for sure. Nevertheless, several articles explain how Russia went to a great extent to interfere with the 2016 Russian election. In 2016, Russia used Facebook and Twitter to interfere with the US election. Using what is known as «Troll farms», professional Russian trolls would take both sides and heat the argument concerning significant issues such as abortion, gun control, and immigration. The tactic was to divide the US, which would be beneficial for Russia. The fact that social media can be «weaponized» by spreading fake news is a concerning problem. If this is something that evolves further into world politics, it is safe to say that we won't have a functioning society.

## 1.2 Problem Description

The task at hand was: "Given a claim or some fake news the application should track its coverage in various social media platforms." Input to the system would be a textual claim such as "Earth is flat", the expected output would be a visualization of analytics of the coverage in social networks such as Twitter and Reddit. Visualize how many users are sharing, liking, retweeting, how deep the message spreads etc. How does it compare to non-fake news about a similar topic? The main goal of this thesis is to visualize the data, the spread of the query, and compare two different queries. The project group was given the option to choose which languages and programs to achieve the given problem description.

## 1.3 Goal

As fake news spreads in large numbers worldwide, there is also an increase in non-profit fact-checking organizations. Sites such as Politifact.com display viral posts and articles and, in turn, label them as false or true; however, Politifact does not show any spread or other analytics. Unlike many other fact checking websites our goal is not to tell the user whether or not a claim is true or false. When it comes to fake fact checking websites we want to contribute by developing a website that allows for users to type in a claim, visualize its spread, check the analytics of it, and compare two different claims.

## 1.4 Use cases for the application

The ideal finished product for this project would be to perform tasks that include tracking the spread of some news on social media and visualizing them. For an average person, the website would be interesting to check different queries and play around with the different features. However, the website would also be useful for a researcher studying the spread of fake news in our society. The researcher would be able to see the top users and posts spread and use the information to draw red lines and find commonalities that contribute to the news being spread. The comparison feature would be helpful for someone who wants to check how some legitimate news is being spread versus how the fake news is being spread. The application would also work for users who want to check the spread of other things not involving fake news, for example, if a brand wants to check the spread of their new promotion or check if the brand is being discussed.

## 1.5 Related Work

The application has taken some inspiration from the website <https://hoaxy.osome.iu.edu/> also known as Hoaxy. Hoaxy is a project from Indiana University where the user can type in a query and choose to search by Twitter or by articles on Twitter.[1] After the user has searched for a query, the website returns a timeline graph that shows the tweets over time. There is also a network graph representing the tweets as nodes; the nodes represent the spread of a tweet where one can see additional retweets of the tweet and replies. The fundamentals of the Hoaxy website are great, but there is a lot of relevant data that could be displayed for the user. Therefore, this application will use the same fundamentals as Hoaxy but it will contain more components to better visualize the spread of a claim. The website will also expand further than Twitter to other social media platforms such as Reddit.

With the growing popularity and impact of social media, a new sub industry called social listening is growing simultaneously. Social listening is the process of monitoring conversations and interactions on social media platforms. [2] Companies like Brandwatch in this industry track different brands and campaigns for other companies. This process can tell the company a lot about how well the company performs on social media, which is extremely important in this day and age. The underlying technology of the application is similar to the social listening platforms. However, instead of collecting data to analyze how well a brand is performing on social media, it aims to use the data to visualize the spread of misinformation.

## 1.6 Deployment

As per now the website is deployed on: <https://ramtinhaf.github.io/>. The front-end and back-end had to be deployed separately because of the Vue and Flask combination. The front-end is deployed via Github and the back-end is deployed with Heroku. Link to the Github repository is provided [3] [4] and the application can be ran locally.

## 1.7 Contributions

Name	Contribution
Awalle	Developing the front-end part of the project(Layout and styling of the website)
Jone	Delveloping the back-end part of the project and creating the side by side comparison
Ramtin	Developing both the back-end and front-end part

**Table 1.1:** Contributions



## 1.8 Outline

The thesis is outlined as follows:

**Chapter 2** introduces the background theory of the current thesis. Revolving around the fake news, related work, different programming languages and libraries.

**Chapter 3** presents how the architecture of the back-end is built up including the formatting of data, models and data visualization.

**Chapter 4** describes how the front-end is designed, the different pages and the components used in the website.

**Chapter 5** is the discussion part where the different problems, challenges and the possibility of further development are discussed. There is also a section for experimental evaluation and a conclusion.



## Chapter 2

# Background

This chapter intends to present the different programming languages and frameworks that were used in this thesis. The first section will be about the spread of fake news on social media, the main programming languages used and their purpose for the thesis. Furthermore, the extension frameworks used will be explained. The end of the chapter describes the Twitter and Reddit platforms and explains what type of analytics the platforms uses.

### 2.1 Fake News

One can define fake news as false or misleading information that are presented as actual news. It often aims to damage the reputation of a person, entity or to make money through advertising revenue. However, the term does not have a fixed definition. The term fake news has been widely used to include false information, including unintentional and unconscious mechanisms, and also by high-profile individuals to apply to any news unfavorable to his/her perspectives.

According to a BuzzFeed analysis the most popular fake news stories from the 2016 U.S presidential election received much more engagement on Facebook than the top stories from the major media outlets. [5] This is a testament to fake news having the ability to reduce the impact of real news.

According to a 2019 study published in Science by MIT Sloan professor Sinan Aral, Deb Roy and Soroush Vosoughi of the MIT Media Lab, false rumors spread faster and wider than true information. [6] In the study the scientists found that fake news were 70 percent more likely to be retweeted on Twitter than truthful news. Also, the fake news

would reach its first 1500 people six times faster than the truthful news. The effect is even more pronounced with political news than other categories.

## 2.2 Programming Languages

A combination of Vue.js, JavaScript, and Python is used to create ow. This combination is a common practice for web development. JavaScript is used to handle the data that is being passed from the back-end, while Vue.js is the front-end JavaScript framework used to build and design the user interface.

### 2.2.1 Python

Python is the chosen language used for creating the back-end part of the application. The back-end uses Python for its framework named Flask. Python is known for its simple syntax and short code length. To make the back-end perform the required tasks, multiple external libraries had to be implemented; examples of these are "TextBlob" and "Geocoder." Python was chosen for the back-end part to handle tasks such as API calls, high-speed data sorting, and parsing the data.

### 2.2.2 HTML

HyperText Markup Language is also known as HTML. HTML is the language used for communication with the web and for creating web pages. Over the last few decades, web programming has evolved and gone through many changes, but the HTML part has been there since the start. HTML is seen as one of the building blocks of web development. [7]

### 2.2.3 CSS

CSS stands for Cascading Style Sheets, which is the language used to style and format web pages. Without any CSS, all websites would be the elements in the HTML displayed downwards. When tags like <font>, and color attributes were added to the HTML 3.2 specification, it created much trouble when creating large web applications. To solve this issue, the World Wide Web Consortium (W3C) created CSS. A good analogy is looking at web development as building a house, the HTML is the foundation, and the CSS is the style/layout of the house. Every house needs a foundation, and it is up to the owner to style the house as he likes. [8]

## 2.2.4 Bootstrap

Bootstrap is an open-source toolkit. It is made for the purpose of designing and developing HTML, CSS and JS. Both CSS and Javascript frameworks are supported. The bootstrap framework makes the styling and placements of components easier.

## 2.2.5 JavaScript

Continuing the analogy about looking at web development as building a house. With HTML and CSS, we have the foundation, the layout, and the style of the house. However, we do not have any functionality in the house like electricity or running water; adding this to the house would be the same as adding JavaScript to a web application. In the application, JavaScript is used to communicate with the back-end to send the query and receive the data that is going to be displayed in the front-end.

## 2.2.6 Flask

Flask is a third-party Python framework for server-side web application development. Using Flask in a web application will provide the user with the tools, technology, and libraries necessary for web development. We have used Flask as our back-end, which handles all of the API calls and the data formatting.

## 2.3 Vue

### 2.3.1 Vue.js

Vue.js also known as Vue, is an open-source front-end JavaScript framework for building user interfaces and single-page applications. In the earlier days of web development, most of the coding was done on the server-side of the web page, but now more of the code is in on the browser side. This is thanks to the addition of JavaScript, but projects involving a lot of JavaScript code could be very messy and complicated to work with. Here is where the JavaScript frameworks are remarkable; They help make the code more organized and easier to use. There are many JavaScript frameworks to choose between, for example, React and Angular. All of the frameworks would work fine with the thesis, but the group decided to use Vue because of experience with the framework. Vue is used to make the website reactive, meaning if any data in the JavaScript file changes, it instantly changes the browser. The Vue framework is used for many different purposes.

Vue is used to create components, route on different pages, and handle each search's different states in the project. The use of components makes the application much more effective; They help extend essential HTML elements to encapsulate reusable code. This means that everything is connected; when routing on a page, the same navigation bar is used everywhere, and Vue knows its exact state using Vuex.

### 2.3.2 Vuex

In the Vue application there is many different components that uses the data from the back-end. When the application grows and more components is added it can become difficult to keep track of all the states. Vuex is Vue's state management pattern which is designed after the Flux pattern designed by Facebook. The Vuex pattern uses a store that stores the state and it is possible to access the store from all of the different components. Vuex is used to store all of the data and fetch the data from the back-end.

### 2.3.3 Chartkick.js

Chartkick is a framework for displaying different charts and graphs. Chartkick is compatible with many different programming languages and especially with Vue. To display a chart in a Vue application it only requires one line of code.

```
1 <line-chart :data="{ 'name': listdata.query, 'data': listdata.linechart }" ></line-chart >
2
```

**Listing 2.1:** Using the Chartkick.js library

With other frameworks, one would need a couple of hundred lines to display the same chart. Using Chartkick, one can choose between the Chart.js library or the Highcharts.js library to display the charts and graphs. The group decided to use Highcharts because the charts looked better than the Chart.js charts, and the charts were reactive. In short, Chartkick is a simplified way of using the Highcart.js library in Vue.

### 2.3.4 D3.js

D3.js is a JavaScript library for manipulating documents based on data. D3 helps bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation. [9] In this application D3.js was used to create a node network.

## 2.4 JSON

JSON short for JavaScript Object Notation which is a lightweight data-interchange format. The JSON text format is language independent, originated from JavaScript, but is now completely independent from JavaScript. It is easy to read a JSON response both for a human and easy to parse and generate for a computer; therefore, it is one of the most used text formats in coding. JSON is mainly used to send data between a server and a web application. In the application, JSON is used to respond to the API calls and send the data from the back-end to the front-end. There are other alternatives to JSON, such as XML or CSV, but JSON seems to be the better alternative for this project. JSON is easier to learn and understand, faster, and since the API's returns the responses in a JSON format, it would make sense to use JSON for the rest of the application.

## 2.5 Twitter

Twitter is a social media platform that works as a microblogging service that allows users to send and read other users' posts are called tweets. Twitter was released on 21. March 2006, and has grown in popularity ever since.[10] In 2020, 500 million tweets were posted every day that converts to 6000 tweets per second. [11] As a user on Twitter, you have a couple of different ways to interact with a tweet. The first and obvious one is to like the post if you find the most likable. If you want to want to share the tweet, you can use the retweet function, which re-posts the tweet, so your followers also see the original tweet. The retweet function is what makes tweets go viral, and when fake news is posted on Twitter, the tweet often gets many retweets quickly, allowing it to spread fast before people realize that it is not a legitimate tweet. There is also the option to reply to the tweet which works as a comment function. Twitter also has a quote function witch is similar to the retweet, but with a quote, the user can also add a quote above the retweet.

## 2.6 Reddit

Reddit is a social news platform that allows users to discuss and vote on content that other users have submitted. The Reddit platform launched 23. June 2005 by Steve Huffman. [12] Reddit has become so popular that there is a subreddit for almost every topic, and therefore, everyone will find something within their interest field. To counter spam and spread of fake news, Reddit implemented a karma score for every user. Suppose the user post only made up posts and spreads misinformation. In that case, other users can downvote the posts, and the user would get a low karma score witch other users can

use to determine the user's credibility. Vice versa, if the user posts well-liked and jovial things, the karma of the user would become positive. If a user has high karma, it is a good indication that the user is not posting fake news.



# Chapter 3

## System Architecture

### 3.1 Introduction

In this chapter, the system architecture is explained by how the data is retrieved from the API to how the data is sent to the front-end. The first sections of this chapter will explain how the Twitter and Reddit API works and how the API's are used to retrieve the data needed for this application. We will also go into detail on how the data is formatted before displaying the data. After the data is retrieved and formatted, it needs to be sent to the front end; therefore, the data flow in the application will be discussed.

A block diagram of how the data flow in the application works can be seen in figure 3.1 below:

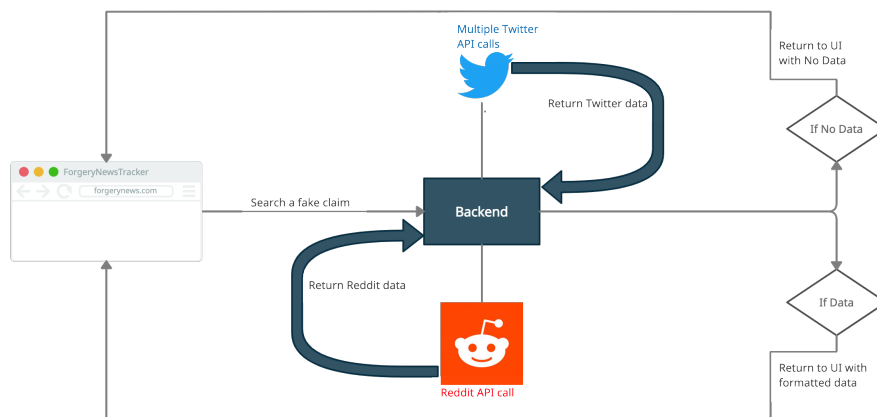


Figure 3.1: Block Diagram

## 3.2 Design Pattern

In order to write flexible, scalable, and reusable code, a design pattern is needed. In object-oriented programming, the SOLID principle is crucial. Relevant principles were used thoroughly when developing the back-end.

The first principle is the single responsibility principle which states that every class should only have one responsibility. In this case of developing the back-end, each function should only have one responsibility. Therefore, there is one function for extracting the data for each component. For example, there is a function for extracting the top users and another for extracting the top posts, this could be done in the same function, but then it would interfere with the principle.

The following principle is the open-closed principle; the principle states any software entity should be open for extension but closed for modification. This means any function in the back-end should not be open for changing the code but open for adding more functionality. If any of the already existing code inside the functions in the back-end is changed it would create problems therefore the code is closed for modification. However, if Twitter were to add a measurement like an amount of how many times a tweet was sent as a message. It would be easy to scale the function to extract the data for this without creating any problems. Having this principle in mind makes the code scalable for later use.

The last principle that was used is the interface segregation principle. This principle states that clients should not be forced to depend upon interfaces that they do not use. In this application, the focus for this principle was to split the different parts up as much as possible, especially for the Twitter and Reddit parts. Instead of combining the two parts as one, the data for each social media is fetched one by one instead of combining them. If the application were to be extended to another social media platform, the interface segregation principle must be kept in mind.

## 3.3 Using the API's for retrieval of data

### 3.3.1 Twitter API

The Twitter API provides the tools needed to retrieve data and analyse the conversations happening on Twitter. In order to use the Twitter API, you need to apply for a developer account. The reason for this being that Twitter wants to know what the API is going to be used for and how you are going to use it. There are two available versions of the

Twitter API, v1 and v2. The application uses the Twitter API v2 because the second version is an improved version of the first. The second version returns a data array in JSON format and the first version returns objects in a statuses array. There is also an improved authentication process improving the security of the API.

In the starting phase after getting the developer account approved, it was conducted experiments with the API using Postman to check what the API returns from Twitter with different parameters. To send a request to the Twitter server the url: "https://api.twitter.com/2/" is used. In order to call the API with a user selected query, the */recent* endpoint is called which accesses the public tweets posted over the last 7 days. A call to the API with the query "CNN is propaganda" will be returned in this format:

---

```
1 {'data': [{'id': '1383002367868727297', 'text': 'RT @Liz_Wheeler:
2 Twitter suspends James O Keefe the week he released BOMBSHELL undercover videos
3 exposing CNNs propaganda & lies. This is '}]}
```

---

**Listing 3.1:** Default Twitter API response

From this default API call, only the ID of the tweet and the text is returned. In order to get more information about the tweet the parameter *tweet.fields* is added to the URL where the parameters *public\_metrics*, *created\_at*, *referenced\_tweets*, *author\_id*, *in\_reply\_to\_user\_id* is added to the *tweet.fields* value.

After adding the *tweet.fields* parameter the API returns the output:

---

```
1 {'data': [{'text': 'RT @Liz_Wheeler: Twitter suspends James O Keefe
2 the week he released BOMBSHELL
3 undercover videos exposing CNNs propaganda & lies.
4 This is..', 'public_metrics': {'retweet_count': 1792,
5 'reply_count': 0, 'like_count': 0, 'quote_count': 0},
6 'id': '1383007692025905153', 'referenced_tweets':
7 [{'type': 'retweeted', 'id': '1382786815992733697'}]},
8 'author_id': '19830815', 'created_at': '2021-04-16T10:41:43.000Z'}]}
```

---

**Listing 3.2:** Twitter API response with parameters

From the output we can see the 'referenced\_tweets' operator is included, inside 'referenced\_tweets' we can check what type of tweet it is. Here the tweet is a retweet of another tweet, other options are replies, quote or an original tweet but the 'referenced\_tweets' operator would not appear for an original tweet. The API also returns data for the date the tweet was posted, the id of the author and the metrics which include the retweets, likes, replies and quotes. From this example output there is 1792 retweets but zero for all of the other metrics. The reason for this is because the Twitter API returns the retweets of the original tweet that was retweeted. In the formatting section it will be explained how the data from the output is extracted to make sure the data displayed is correct.

So far all of the data about the tweets has been returned by the API, but the */recent* endpoint does not give any data about the user tweeting except for the *author\_id*. To extract the data about the users all of the author ids need to be extracted and by using the */users* endpoint the API will return the username, location, public metrics for the user and the profile picture. Below is a snippet of how the data is returned:

```
{'data': [{'id': '19830815', 'username': 'gasmith2', 'name':
'Freedom #ShallNotBeInfringed', 'location': 'USA First',
'public_metrics': {'followers_count': 347, 'following_count': 1996,
'tweet_count': 10924, 'listed_count': 0}, 'profile_image_url':
'https://pbs.twimg.com/profile_images/131873376542755848/tHUBEqbc_normal.jpg' ]}]}
```

**Listing 3.3:** Data returned for a user

Combining the */recent/* and */users* API calls, there is enough data to display all of the data at the front-end.

In order to call the Twitter API without getting any errors the query needs to be URL encoded also known as percent-encoding. URL encoding a query is parsing the query only using the US-ASCII characters, for example when the user is searching for "#bitcoin" the API would be called with the query "%23bitcoin". Using the URL encoded query eliminates the errors that occurs when the API is called with a unencoded symbol. To URL encode the query the python library *urllib* is used. The query is URL encoded with a single line of code:

```
1 query = urllib.parse.quote(unencoded_query)
```

**Listing 3.4:** URL encoding

### 3.3.2 Reddit API

In order to use the Reddit API there is not as much formalities as the Twitter API to get access. Reddit only requires a user and the user has to create an application which generates two secret tokens that is required to access the API. To send a call to the API the request package in python is used. A call would look like this:

```
1 import requests
2 res = requests.get("https://oauth.reddit.com/r/python/hot",
3                   headers=headers)
```

**Listing 3.5:** Directly accessing the Reddit API

Accessing the API directly works fine, but the process is a bit confusing especially when trying to extract as much data as possible. In response to developers thinking the Reddit API was confusing the wrapper PRAW was created. PRAW which stands for Python Reddit API Wrapper aims to make the API more accessible and easier to understand. PRAW started as a Github project called `reddit_api` created by Timothy Mellor in 2010.[13]

In order to use the PRAW extension, `praw` need to be pip installed. After `praw` is installed and imported, a Reddit instance needs to be created to access the API. In the Reddit instance the secret tokens are used to authenticate the user.

---

```
1 reddit = praw.Reddit(  
2     client_id=config.client_id,  
3     client_secret=config.client_secret,  
4     user_agent="my user agent")
```

---

**Listing 3.6:** Example of a Reddit instance:

After the Reddit instance is created the API is called with a single line of code:

---

```
1 reddit.subreddit("all").search(query, limit=100)
```

---

**Listing 3.7:** Calling the Reddit API

With the line above the Reddit API is called and returns a maximum of a hundred posts. The parameter within the `subreddit` decides which subreddit to search within. Using the parameter `"all"` calls the Reddit API for every subreddit and returns the top hundred posts for the selected query. The query does not need to be URL encoded for the Reddit API since the PRAW extension handles the URL encoding internally. Any additional API calls with the same query would return the same posts, so pagination does not work for Reddit. Each of the posts is returned a a submission object where all of the data about the post and the user is stored.

---

```
reddit_data = []  
for submission in reddit.subreddit("all").search(query, limit=100):  
    try:  
        reddit_data.append({"author": str(submission.author.name),  
                            "title": str(submission.title),  
                            "upvote_ratio": submission.upvote_ratio,  
                            "upvotes": submission.ups,"url": str(submission.permalink),  
                            "created_at": str(submission.created_utc),  
                            "subreddit": str(submission.subreddit),  
                            "number_of_comments": str(submission.num_comments),  
                            "post_karma": submission.author.link_karma,  
                            "comment_karma": submission.author.comment_karma,  
                            "icon_img": submission.author.icon_img})
```

---

```
except:
    print("User suspended")
```

**Listing 3.8:** Calling the Reddit API

Above is a code snippet of how the data is extracted from the API call. For each submission there is a "Redditor" instance which stores all of the data for the author of each post, within this instance the username, karma of the user and profile picture is extracted. The title, subreddit, upvotes, ratio of upvotes and the date the post was created is also extracted and forwarded to be formatted before being sent to the front-end.

## 3.4 Back-end

The back-end part of the application is deployed separately in works in a similar manner as an API. From the UI the user writes a query or selects a predefined fake claim. The query is then sent to the back-end, and in turn the back-end returns all of the data in the predefined format for the front-end to display.

### 3.4.1 Flask Back-end

The Flask framework was used for the back-end part because of previous experience with using Flask as the back-end. The back-end architecture is designed in such a way that it fetches, formats and sends the data to the front-end in an orderly manner. The main parts of the back-end that handles most of the logic is the *twitter\_search* and the *reddit\_search* methods:

```
@app.route('/twitter/search', methods=['GET', 'POST'])
def twitter_search():
    d = request.json
    ...
    return json.dumps(twitter_data)
```

**Listing 3.9:** Method for fetching Twitter data

```
@app.route('/reddit/search', methods=['GET', 'POST'])
def reddit_search():
    d = request.json
```

```
...  
return json.dumps(reddit_data)
```

**Listing 3.10:** Method for fetching Reddit data

Within these functions the query from the UI is fetched, both of the API's are called, the data returned from the API's are formatted and the formatted data is sent to the front-end using the `/search` route.

From the Twitter API section, the code snippet included a returned tweet from the API; the entry was a retweet of another tweet. The entry returned the number of retweets from the original tweet, but not the other information about the original tweet, such as the likes and replies. Sometimes the API will return some of the retweets of a popular tweet, but not the original tweet.

To counter this issue, the function `extract_retweet_ids` is used where all of the returned tweets of the type "retweet" are extracted. The ID of the original tweet is used to call the API an additional time. The data returned from the additional API call is added to the response from the original API call. Implementing this API call, more tweets are added to the response, and the data is more accurate because the analytics from the original tweet that was retweeted is added.

The reason for this implementation of the back-end, is to make the application easy to scale. When all the data is processed and formatted correctly in the back-end, there is minimal implementation that needs to be done to display the data in the front-end.

### 3.4.2 Pagination

Since the Twitter API can only retrieve one hundred tweets per call, pagination was implemented to retrieve more data from the API. The first method used to get the most amount of tweets was to use a timer and call to the endpoint multiple times with a two second delay between each call. This method would return many duplicates, and the performance was slow because it has to wait a specific amount of time for each call. To improve this, by using the `start_time` and `next_token`, actual pagination is achieved. This is because the `next_token` will give us the page of the tweets that were not retrieved. [14]

The implementation of the pagination is listed in the code snippet below:

```
def get_tweets(query, headers):
    # Rewind one week
    start_time = datetime.utcnow() - timedelta(weeks=1)

    # Header must be in iso format
    start_time = start_time.replace(microsecond=0).isoformat() + "Z"

    tweets = {}

    search_url = create_recent_search_url(query, start_time, None)

    response = twitter_recent_search(search_url, headers)

    for i in response.get("data", []):
        tweets[i["id"]] = i

    # max result of tweets retrieved can be set here(now 500):
    while response.get("meta", {}).get("next_token") != None and len(tweets) < 500:
        search_url = create_recent_search_url(query, None,
        response["meta"]["next_token"])
        response = twitter_recent_search(search_url, headers)
        for i in response.get("data", []):
            tweets[i["id"]] = i
    return tweets
```

**Listing 3.11:** Pagination implementation

Each time a request is sent to the endpoint called the Twitter API recent search endpoint [14], it will return a meta including "next\_token" telling if there are other pages that have not been returned in the last API call. Next\_token is a hash that is sent with the following API call. Therefore we are looping through the responses until there are no more pages. There is a set limit for the maximum amount of tweets set to 500 tweets per search. Setting a limit allows the application to have a minimum of 1000 searches per month since the monthly tweet cap usage for our Twitter developer account is 500.000 tweets per month. Setting a limit to a maximum of 500 tweets per search was the best option for performance and visualizing the spread. More tweets would make the user experience worse because of the long waiting time, and with 500 tweets, the spread is nicely visualized.



### 3.4.3 Error Catching

In the Reddit section, the code snippet for extracting the data needed has a try/except statement. The try statement states that for each submission the data is extracted from the submission and added to a dictionary which is stored in a list for all of the Reddit data. If there is an error occurring while extracting the data from the submission, the script excludes this submission. The only error message that appears is when a user is suspended from Reddit. This means that the user does not have a karma score.

The main issue that resulted in many errors was that the query used to call the API would not return any data. The reason the API does not return any data could be that the search is too specific or it could be that it is not relevant. To fix this problem a check had to be implemented to check if any data has been returned before any functions are called. If the query does not return any data the JSON response that is being sent to the back-end is changed to:

---

```
1 json_response = {"data": "No data"}
```

---

**Listing 3.12:** JSON response for no data

When the data is sent to the Vuex store there is a check if the data is set to "No data". If there is no data the property data is set to false. Implementing this eliminates the errors that occur in the front-end when components are trying to access properties that does not exist when the API returns nothing.

Both of the API calls has a try/except statement to catch any unknown errors in the back-end to prevent any unwanted errors in the front-end.

## 3.5 Formatting the data

When the data is returned from the API's the data is returned as a list containing multiple dictionaries. These dictionaries contain the results. In order to display the data returned, it needs to be formatted. For each component shown in the front-end there is a function to extract the exact data that is needed for the component. An example would be; the bar chart for Twitter, the function `create_barchart` is called with the JSON response which is the unformatted data. The function extracts the likes, retweets, replies and quotes, returns the combined analytics as a list which is sent to the front-end to be displayed. This approach is used for every component and if there is another component that is going to be added to the website it is easy to extend the back-end to format the data for another component.

### 3.5.1 Formatting Twitter data

The data returned from the Twitter API is in the format of a list with a dictionary for each entry/tweet. The procedure for extracting and formatting the data is very similar for each Twitter component. The main difference between the procedures is the type of data being extracted. Continuing with the example where the data for the bar chart is being extracted and formatted can be seen in the snippet below:

```
def create_barchart(tweets):
    total_retweets = 0
    total_likes = 0
    total_replies = 0
    total_quotes = 0

    for id_, data in tweets.items():
        if "referenced_tweets" in data:
            if data['referenced_tweets'][0]["type"] != "retweeted":
                total_retweets += data['public_metrics']['retweet_count']
                total_likes += data['public_metrics']['like_count']
                total_replies += data['public_metrics']['reply_count']
                total_quotes += data['public_metrics']['quote_count']
            else:
                total_retweets += data['public_metrics']['retweet_count']
                total_likes += data['public_metrics']['like_count']
                total_quotes += data['public_metrics']['quote_count']
                total_replies += data['public_metrics']['reply_count']

    barchartlist = [['Likes', total_likes], ['Retweeets', total_retweets],
                   ['Replies', total_replies], ['Quotes', total_quotes]]

    return barchartlist
```

**Listing 3.13:** Formatting the data for the bar chart

The methods use a for loop to access every entry and extract the likes, retweets, replies and quotes. Referring to the 3.2.1 chapter, the returned data gave the retweet count of the original tweet. To make sure that this retweet number does not get counted multiple times, we check to see if the tweet type is retweeted. If the tweet type is in fact retweeted, it is not counted. The same procedure is used for the line chart where the dates of the tweets are extracted and formatted in a dictionary for the line chart component or for any other component for the Twitter part of the application.

### 3.5.2 Formatting Reddit data

The process of formatting the Reddit data for the components in the Reddit view is very similar to the Twitter process because the data returned from the APIs are in the same format. Both procedures involve a for loop to extract the data from each entry; the most significant difference is the type of data extracted. The two platforms use different types of analytics. Below is an example of the data for the engagement component being extracted:

```
def reddit_engagement(reddit_data):
    engagement = {}
    user_ids = []
    upvotes = 0

    for i in range(len(reddit_data)):
        upvotes += reddit_data[i]["upvotes"]
        if reddit_data[i]["author"] not in user_ids:
            user_ids.append(reddit_data[i]["author"])

    engagement["posts"] = len(reddit_data)
    engagement["users"] = len(user_ids)
    engagement["engagement"] = upvotes

    return engagement
```

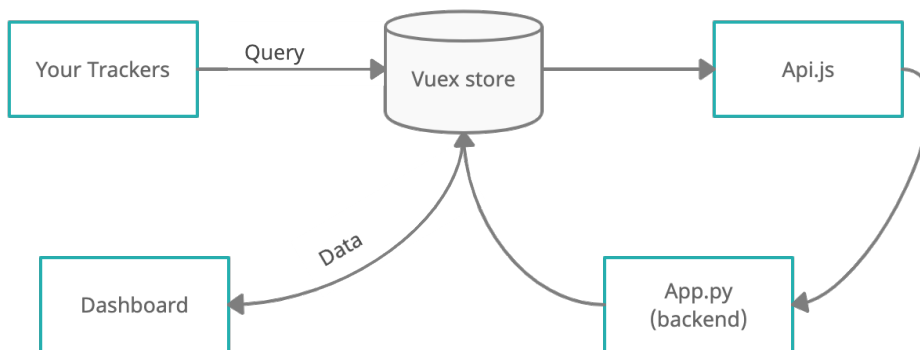
**Listing 3.14:** Formatting the data for the engagement component

The method adds each unique user to a list and counts the total users, posts and upvotes and returns the totals in a dictionary. This procedure is repeated for the Reddit components with slight modifications that rely on what format the data is needed to be displayed in the front-end.

## 3.6 Communication between back-end and front-end

To create a full stack application the back-end and the front-end needs to be able to communicate. The front-end needs to be able to send the query the user chooses and send it to the back-end so the back-end can call the API's with the chosen query. After the data is formatted the data needs to be sent back to the front-end where it is received in the Vuex store. To achieve these task the request library is used in the back-end and in the front-end the promise based HTTP client Axios is used.

The process begins by calling the `getResults` function, when the user clicks the "Add search" button in the front-end, inside the `getResults` function an instance of the "BackendApi" class is made. Inside the class the function `getMessages` is called which make a post request using Axios, the route `/showinfo` is then used to send the query to the back-end. In the `showinfo` method in Python the POST request from the front-end is retrieved, the API's are called and the data is formatted. When the data is ready to be sent back the back-end uses a POST request to the front-end where the data is retrieved in the store and is ready to be displayed.



**Figure 3.2:** Flowchart on how the different scripts communicate with eachother.

How a query is retrieved and furthermore how the Twitter data is sent to the front-end is displayed in the code below, this is done in the same manner for Reddit.

```

@app.route('/twitter/search', methods=['GET', 'POST'])
def twitter_search():
    d = request.json
    ...
    return json.dumps(twitter_data)
  
```

**Listing 3.15:** How the query is fetched and the data is sent to the front-end

Sending the query and get the data in the Vue store:

```

async getResult(state, searchValue) {
  try{
    let api = new Backendapi();
    let twitter_response = await api.getMessages_twitter(searchValue);
    console.log(twitter_response)
  }
}
  
```

**Listing 3.16:** How the data is retrived in the front-end

```
import axios from 'axios';

export default class Backendapi {
  getMessages_twitter(query){

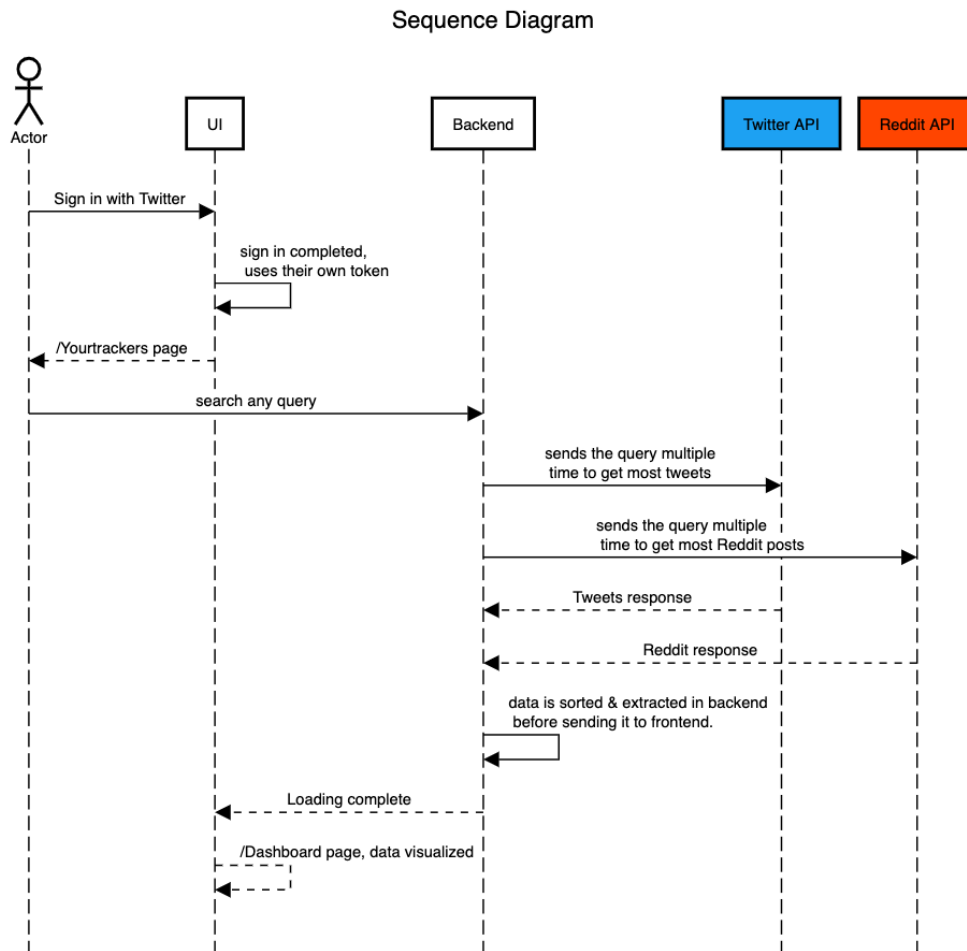
    // create connection with the results via Heroku

    const path = 'https://fnt-backend.herokuapp.com/twitter/search';

    return axios.post(path, {query}).then((res) => {
      return res.data
    })
  }
}
```

**Listing 3.17:** How the front-end communicates with the back-end

### 3.7 Sequence Diagram



**Figure 3.3:** Sequence Diagram

A sequence diagram shows how the operations are carried out in the application. It shows the interactions between object instances. In our application the sequence is like this:

- The user uses their own twitter Token to do the searches. So they will sign in using Twitter.
- Redirect to /yourtrackers page. Here the user can search on whatever the application should visualize. When clicking the "Add Search" button, an API call will be made.
- From the Back-end it will request posts from both Twitter API v2 and Reddit API.
- The response from the APIs will be in JSON format. The data will be extracted and sorted so that when it is sent to the front-end everything is ready to be displayed.
- On the /yourtrackes page, a check icon will be represented beside the search which indicates that the search is ready to be displayed on /Dashboard.

## Chapter 4

# Front-End Design

### 4.1 Introduction

A good web page is user-friendly, easy to navigate, and intuitive. To achieve these things, the developers must keep this in mind throughout the whole development. A good design plan starts with a good skeleton and requires a focus on scalability from the beginning. This chapter contains an explanation and review of the website's layout, design choices, and front-end components.

### 4.2 Pages and design

The website consists of five pages. There is the Home/Landing page, "Your Trackers", "Dashboard", and three info pages with "About us" and "FAQ". The reason for choosing the layout/website design that we have is because it is supposed to serve as an easy website to navigate in, and thus we have very few pages as well as we want it to serve as very intuitive and easy to understand.

The website has a simple design with a focus on ease of use, and it serves as a website where one can easily retrieve information on the selected search. It has few pages, tabs, and links to each section, making it easy for the user to navigate the website. There was also a focus on designing the website, so it was good-looking and had a professional look to it.

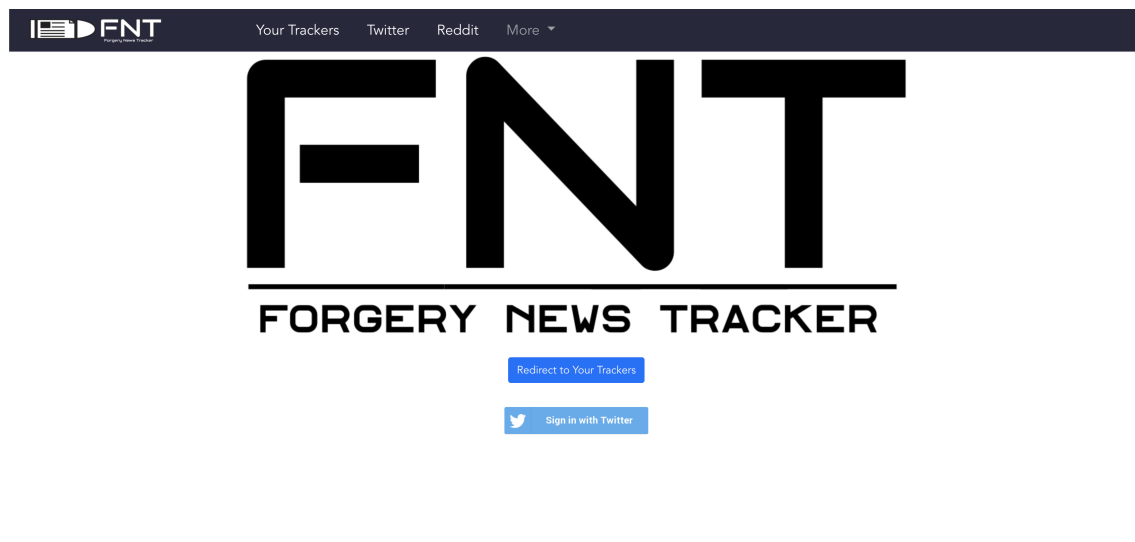
The design phase began initially with a sketch to decide the page count, what components to add, where to locate the different components of the front-end, and an initial color palette. The constant parts such as the header and footer were initially designed, then the individual pages of the site was made.

### 4.2.1 Landing Page

In the landing page there is a picture of our logo along with two buttons which gives the user the opportunity to redirect to the trackers page or sign in with Twitter.

When designing the landing page, there were two things in focus—having the logo visible and redirecting directly to the page called "Your Trackers". The Sign-in with the Twitter button was implemented to have the users authenticate themselves. When signing in, the user will be redirected to the Twitter OAuth page to sign in with their account. Then a PIN is displayed on the page, so the user must copy this PIN and paste it into our application. If the PIN code is as expected, the user-tokens will be returned. With these tokens, the application can become more personalized for the user, such as welcoming them with their username and profile picture etc.; furthermore, it could be possible to store searches for each user.

Their token can also be used to make the different searches on the Twitter API endpoint, but it was decided to use the bearer token from the application. The applications bearer token has a greater rate-limit per 15 min and per month versus the users token[15] and is, therefore, a better option. There is the option to use the Twitter login but as per now the version uses the implementation of the applications bearer token.



**Figure 4.1:** Landing page



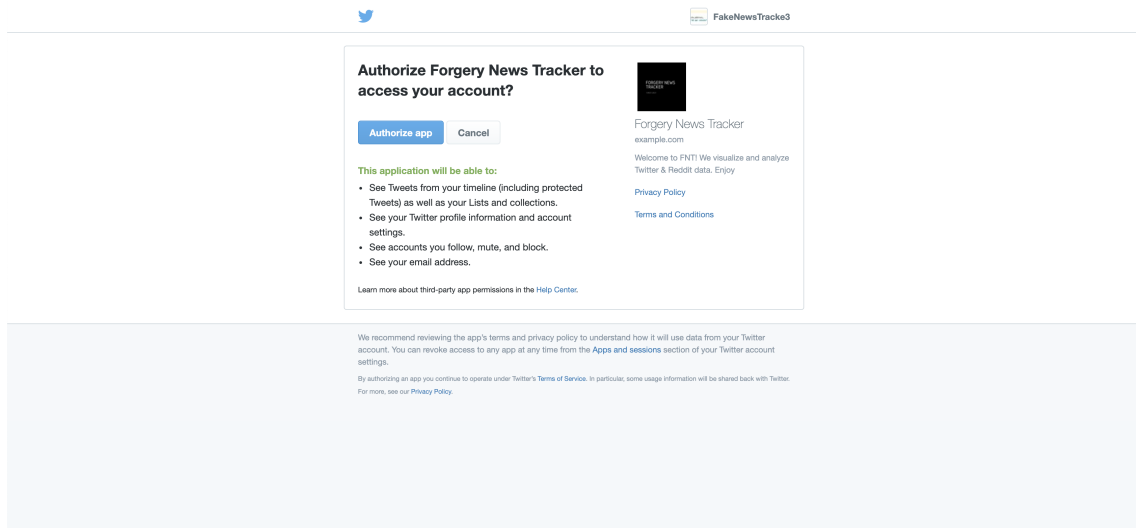


Figure 4.2: Authentication page

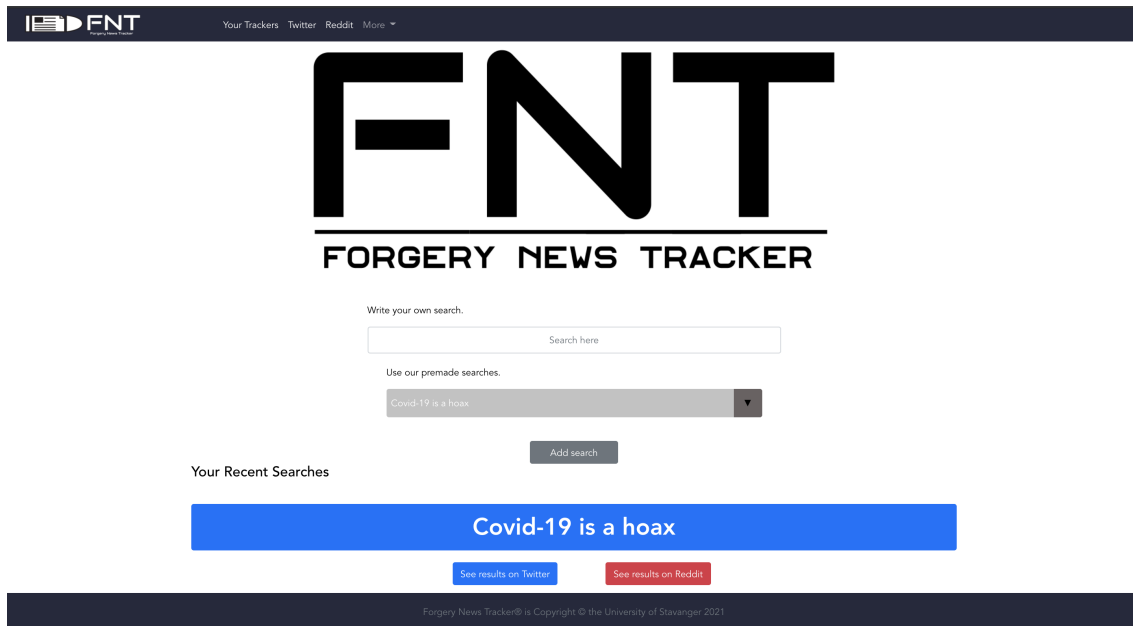
## 4.2.2 Your trackers

In the trackers page you're first met with a greeting message and a short info-text which tells you what to do. You have the opportunity to select already made search queries or add your own queries. When these are added and loaded you get the opportunity to check either one or two of the added queries by selecting them. After this you can press the button that says "See Twitter Results" and "See Reddit Results". This button will redirect you to the dashboard.

The intent of the website is that one wants to search a query and then search for the coverage of this query on various social medias. The your trackers page is the page that makes it possible to specify the query. When designing this page, three things came into consideration: the amount of components, layout of the page and placement of the different components in order to make it as user-friendly as possible.

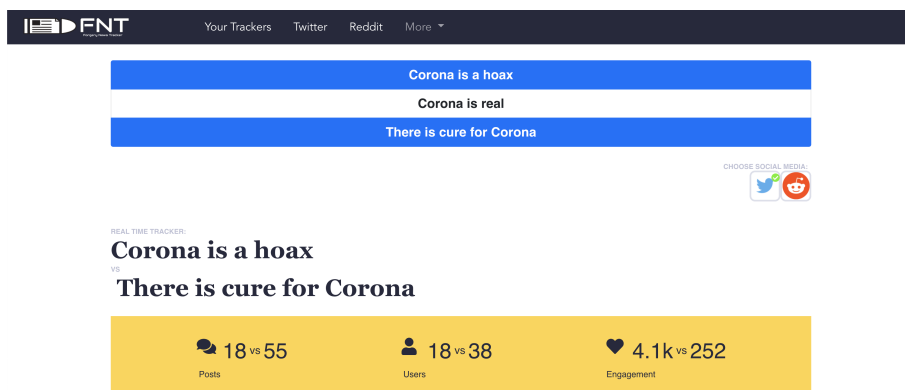
## 4.2.3 Dashboard

In the dashboard pages the queries you have chosen in the trackers page are displayed. If you choose only one query, it will show results only for this query. If you choose two queries on the other hand, the page layout changes and you have a dual-view of the two queries, where you have the same type of visualizations that are set beside each other to compare these. The dashboard page shows the social media engagement of the different queries in a timeline as a line chart, different types of engagement in a bar chart, the spread of a claim between users in a node network as well as tables of the top posts and most influential users writing about the selected search or searches.



**Figure 4.3:** Your Trackers

When designing the dashboard page three things had to be taken into consideration: the sizing of the components, how we wanted to display the data and in which order we wanted to display it. In the dashboard page the different components are displayed so that the user can scroll down and see the different components. The components containing the data are designed and added so that they display information to the user when hovering over. Tool tips with information about the data being displayed are also added on above each of the components displaying data, all of this is done to make the page informative and easy to interact with. The selected queries are put at the top of the page and the page allows for users to toggle between the searches or show two searches simultaneously. The design of the dashboard page is altered when two searches are selected. When there is a dual view every data-component has the same size and are placed aside each other, this is done so that the user easily can compare the two views.



**Figure 4.4:** Dashboard page

## 4.3 Components

Components are a really important feature in Vue.js which make it a lot easier to create scalable and reactive web applications. Components lets the developer focus on one thing at a time as well as it divides the user interface into smaller reusable pieces which one also can encapsulate into each other. This is a feature which makes it a lot easier to create scalable and reactive web applications and is also the reason as to why this approach was chosen.

The website is made up of a lot of different components. the website itself is one big component, the different pages are each their own component and also the different pages have components within themselves. Considering the fact that most of the pages does not have a lot of different elements and information, most of the pages are one-component pages which doesn't encapsulate other components. The dashboard page (which is a component in itself) on the other hand consists of a range of other components. The page consists of a search list, tracker header, engagement bar, line and bar chart, node-network and geo chart etc. All of these are separate components which have been made on their own, and then placed in the dashboard component.

```
<template>
  <div class="container">

    <SearchList/>
    <Trackerheader class="dashboard-comp"
      :listdata='Display1.query '
      :listdata2='Display2.query' />

    <Engagement class="dashboard-comp"
      :listdata1='Display1 '
      :listdata2='Display2' />

    <div class="container_for_linechart">
      <LineChart class="dashboard-comp" id="linechart"
        :listdata1='Display1 '
        :listdata2='Display2 '
        v-show="Display1.query && Display2.query !== {}"/>
    </div>
    ...
  </div>
</template>

export default createStore({
  <script>
  //here we import other components
  import BarChart from '../components/BarChart '
  import BarChartBig from '../components/BarChartBig '
  import LineChart from '../components/LineChartComponent '
  import SearchList from '../components/SearchList.vue';
  import topPosts from '../components/Topposts';
  ...
})
```

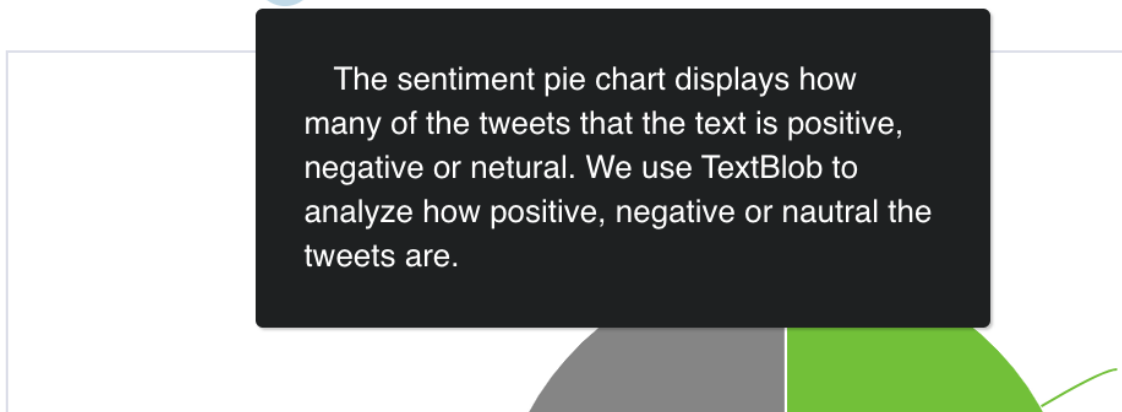
```
export default {
  name: 'Dashboard',
  components: {
    BarChart,
    BarChartBig,
    LineChart,
    SearchList,
    topPosts,
    ...
  }...
}
```

**Listing 4.1:** Importing the different components into the dashboard component

### 4.3.1 Tooltip for Components

Each component that visualizes data has a tooltip that explains how it works when hovering over the question mark as shown in the figure below. This is used if the user does not understand the chart or just wants an explanation of data data is displayed.

## Sentiment ?



**Figure 4.5:** Tooltip component

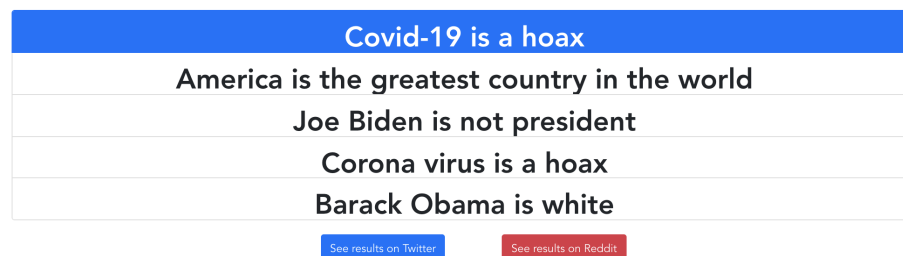
### 4.3.2 Search list

The search list is the component where the queries are stored when the user has added the search. The queries are displayed in a table where the user can click an entry; when the user clicks, the query is activated and displayed. If a query is activated, the selecting row turns blue.

To keep track of which of the queries are displayed, the "searches" list in the store contains objects with the keys: query, active, loaded, and index. When clicking a row in the search list, adding the query to the display list is called. Here it is checked if the query matches the query in the "allTweets" list; if it is the same, the query is displayed and the index in the displayed list is set. When the user deselects a query, we use the index property in the "searchlist" and remove the item displayed at the particular index. The search list is used both on the tracker page and on the dashboard page.

```
[language=JavaScript, caption=Setting tweet to the displayed list]
DisplayTweet(state, idx){
  for (let i in state.allTweets){
    if (state.searches[idx]["title"] == state.allTweets[i]["query"]){
      state.tweets.push(state.allTweets[i]);
    }
  }
  state.searches[idx].active = true
  state.searches[idx].index = state.tweets.length -1
}
```

Your Recent Searches



**Figure 4.6:** Searchlist component

### 4.3.3 Engagement

The engagement component is the yellow bar displayed in the top section of the page. This component displays the social media engagement in the form of amount of posts, users writing about the specific query or queries and engagement which in this context is referred to as total retweets and likes of posts containing the specific query or queries.

## There is cure for Corona

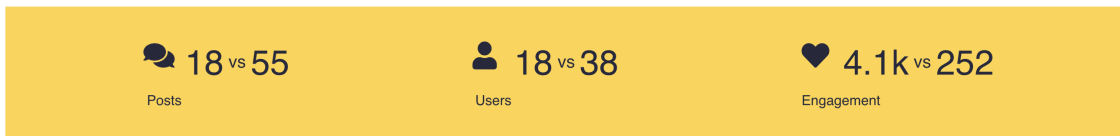


Figure 4.7: Engagement component

### 4.3.4 Line chart

The line chart is a time line that shows total cumulative tweets including retweets for the query over the span of a certain time period, which is ideally supposed to be seven days, but if the query is very popular it can be as short as one day. This is because it is retrieved the last one hundred tweets from the twitter API and thus, if it is a very popular query the last one hundred tweets won't necessary be from the last seven days. By displaying this timeline in the form of a line chart we are able to display total engagement in the form of tweets, retweets, quote tweets and replies as cumulative tweets.

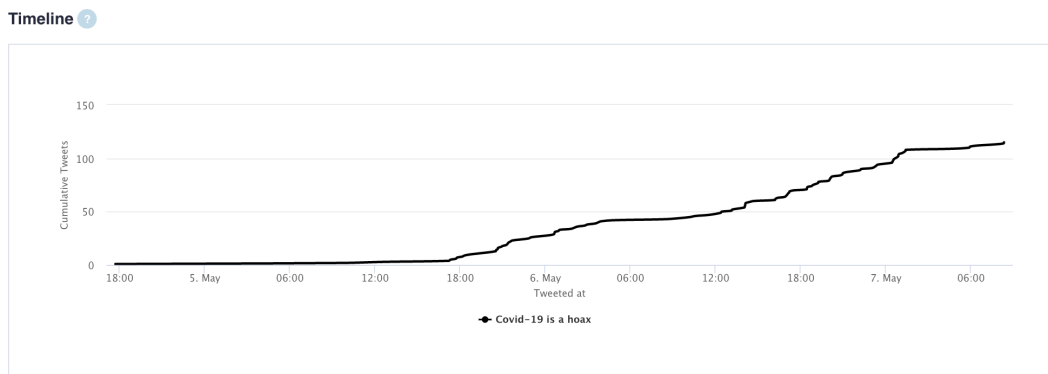


Figure 4.8: Linechart component

### 4.3.5 Bar chart

The bar chart is a visualization of the different types of social media engagement. It shows total amount of likes, retweets, replies and quotes on the different tweets containing the query. When you are comparing a truthful claim versus a fake claim this is a good indicator to see how much a fake claim is spread compared to a truthful one.

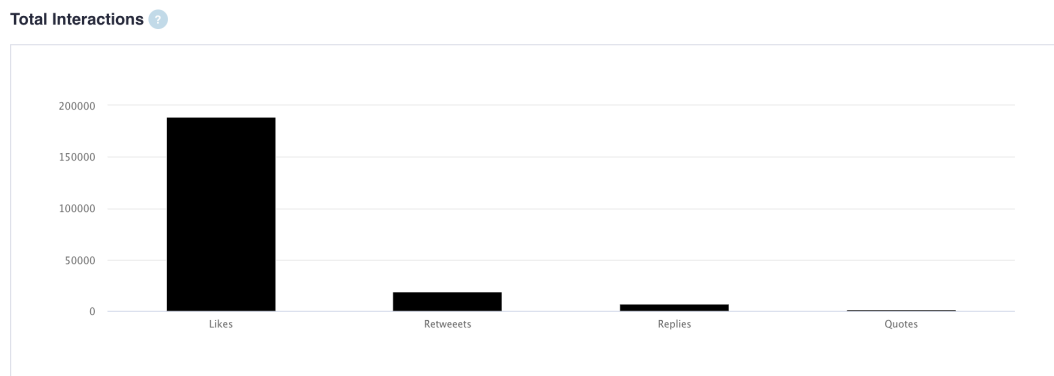


Figure 4.9: Bar chart component

### 4.3.6 Geo chart

The geo chart component shows the country from which the twitter users are tweeting from. Essentially this is supposed to give an indication on in which locations certain queries are tweeted the most about, which can be useful in many different ways in relation to the spread of fake news. When calling the API there is a parameter for the tweet fields called *geo* which returns the location of where the tweet was tweeted from, but to get access to this functionality you must have the premium API. To create a geo chart the locations is set by using the user provided location from their Twitter bio. This could be anywhere the users wants and thus this graph is not necessarily as accurate and helpful, but in a further developed website this would be very insightful.

Accurate geocoding results are an essential part of many geospatial processes. Whether you want to show your retail locations on a map, calculate an optimized route for a delivery, or search within the radius of an origin point, the geocoding API enables you to associate latitude and longitude with an associated address. [16]

Another issue with this is that a lot of the users often write cities instead of country in their bio, and the chart library from Google Maps used for the geo charts only accepts country codes. To solve this problem an API called MapQuest was used[17]. Using the Geocoder library it is possible to access the MapQuest API easily with a single line of code:

---

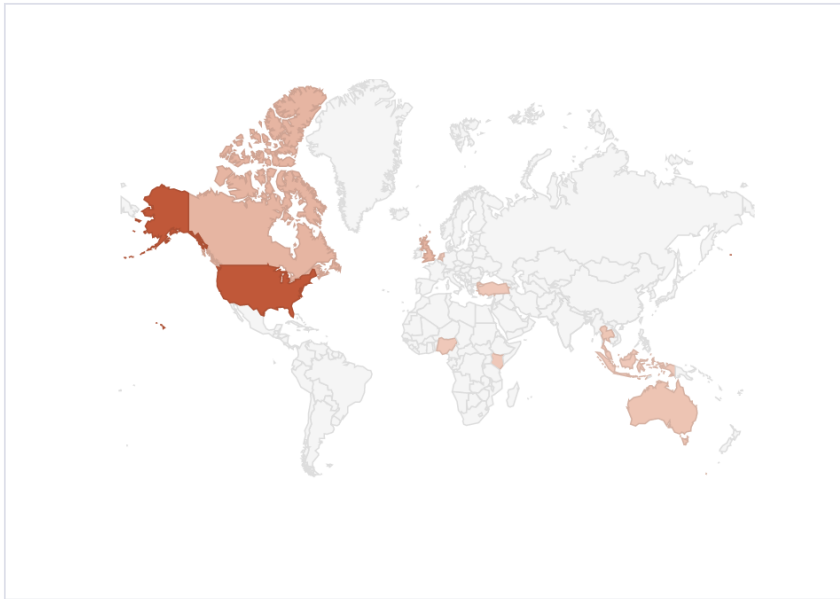
```
1 g = geocoder.mapquest(all_locations, method='batch',key="X")
```

---

**Listing 4.2:** Calling the Mapquest API with Geocoder

With this API call the website is able to retrieve a list with all of the locations and then passes the data to the chart library. The format of country code and an integer for how many entries there was from the specific country.

#### Locations ?



**Figure 4.10:** Geochart component

### 4.3.7 Node network

To show the interaction between users and the spread among different users the node network was created. In the node network component, every user that has tweeted or retweeted any tweet about the query are shown as nodes. These are further connected to other users that has retweeted their tweets or retweets including the specific query. To create the node network the d3.js chart library was used. To create the nodes and the links between, we had to first extract them in the correct format from the Twitter API. The links between nodes were extracted as a list of objects in the form of "*{source:a, target:b}, {source:c, target:d}, {source:e, target:f},...*", the nodes were extracted as a list of objects in the form of "*{id:x1}, {id:x2}, {id:x3},...*"



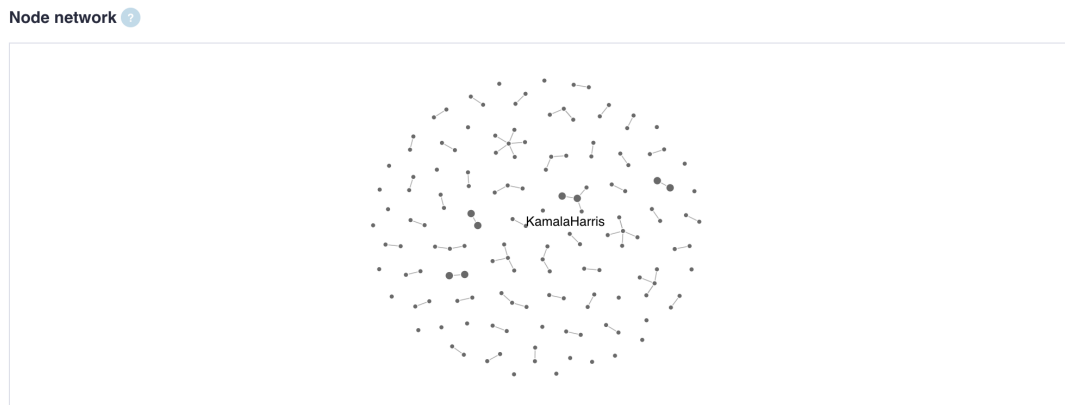


Figure 4.11: Node network component

### 4.3.8 Top Posts and Most Influential Users

There are two components that displays the top posts that the most retweets and the most influential users. The component is in a container that consists of three rows with a Twitter post along with username, profile picture and the amount of retweets and likes on the post. Each of the posts are clickable links which redirects you to the actual tweet on Twitter.

The component with the most influential users shows the users with the biggest following that are tweeting about the query or interacts with a tweet. The component is also a container that contains three rows, on each row there are three smaller containers that contain the Twitter username along with the users following count. Also, each of the containers are clickable links that redirect you to the Twitter page of the actual user.

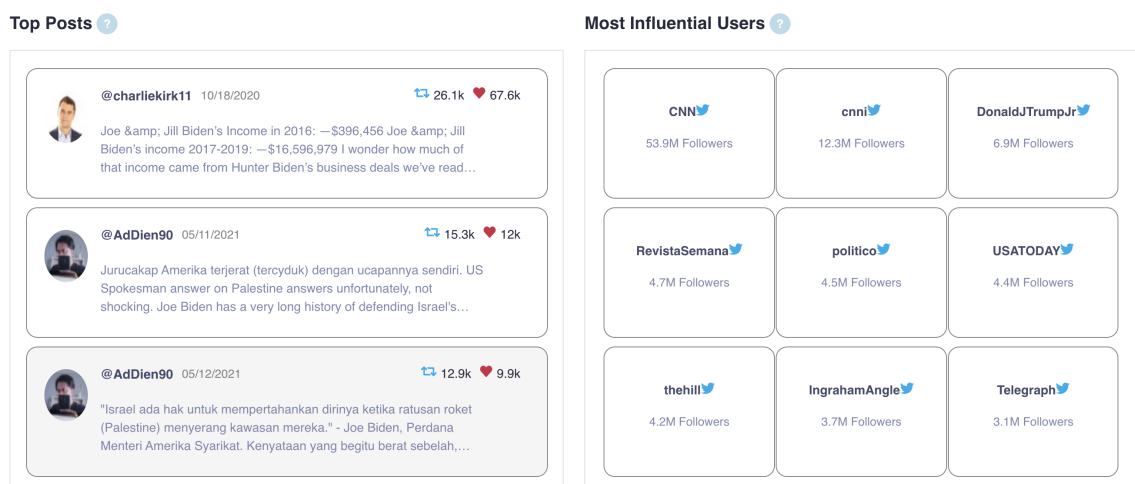


Figure 4.12: Top posts and most influential users components

### 4.3.9 Reddit Word Cloud

The word cloud is a component that displays different words with distinct size in a cloud. The word symbolizes the subreddit and the size symbolizes how many of the posts from the subreddit. The user can click any of the words to be redirected to the selected subreddit on reddit.com. Unfortunately there was not a library for creating a word cloud that was compatible with Vue 3. Therefore the word cloud is created using only HTML and CSS, the word cloud is an unordered list with all of the decoration removed and by using the flex display and flex-wrap properties in CSS the cloud is created. The size of the word is predetermined in the back-end by counting the sum of each entry from the subreddit.

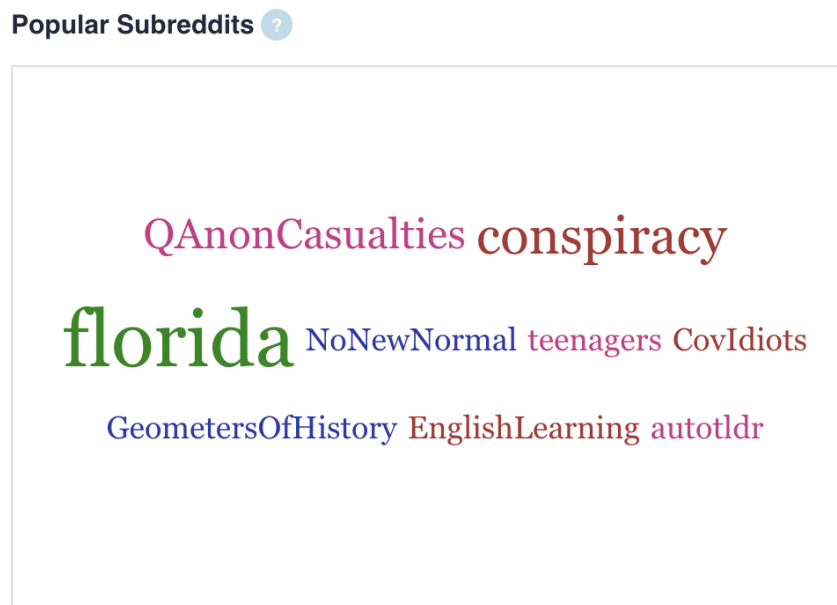


Figure 4.13: Wordcloud component

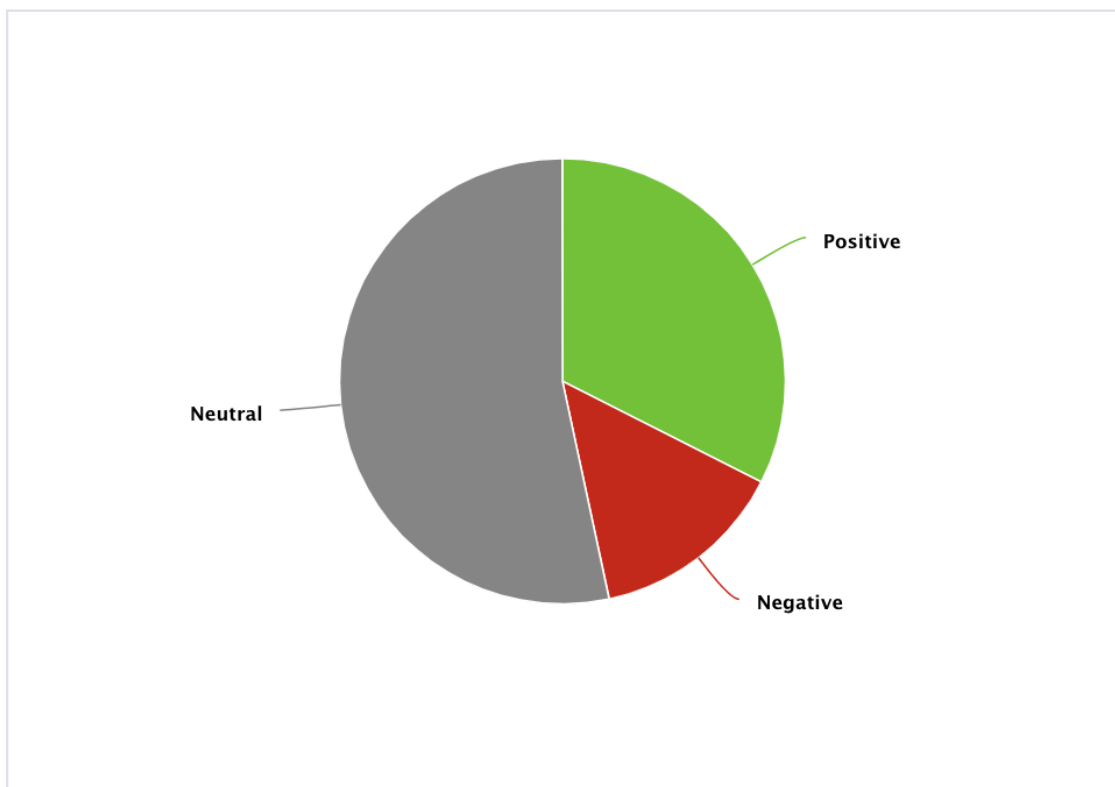
### 4.3.10 Twitter Sentiment Component

The Twitter sentiment component is responsible for measuring the posts sentiment. It is a way to evaluate spoken or written language to determine if the expression is favorable (positive), unfavorable (negative), or neutral, and to what degree. The Twitter sentiment are presented in a pie chart, showing the user how many posts were positive, negative or natural.

When creating the Twitter sentiment the data retrieved from the Twitter API call is first removed of all characters that are not plain text, i.e emojis, symbols etc. To do the sentiment analysis of the tweets we used a python library called TextBlob, TextBlob

uses an NLTK pattern library to analyze each word in a sentence individually. The library allows for sentiment analysis by using built in functions to calculate polarity and subjectivity. Polarity has a score range of [-1.0 to 1.0]. The polarity score denotes the amount of positive or negative information which is presented in the text. A score of -1 is the most negative and the score of +1 is most positive. Subjectivity has a score range of [0.0 to 1.1]. The subjective score helps in understanding the objectivity of the text. A score of 0 is a fact and a score of +1 is very much an opinion. After the polarity and subjectivity scores are found, they are then passed to a function in the back-end that decides the sentiment based on the polarity and subjectivity scores that the built in TextBlob functions returns for each tweet. If the polarity score is  $> 0$  the analysis will be positive. If  $= 0$ , the analysis will be neutral. If  $< 0$ , it will show negative. Lastly, the sentiment of each of the tweets are then sent as a dictionary in the form of "positive": x, "negative": x, "neutral":x along with the rest of the data to the front-end.

## Sentiment ?



**Figure 4.14:** Twitter sentiment component

### 4.3.11 Accessing the data in the components

In the initial implementation there was used a unique state in the vuex store for each component. This became lousy coding practice because it became redundant, and it was not easy to scale, nor was it dynamic. Another thing that made this a bad choice of coding is that as the component count grows, the list count would also grow where each of the lists would have to be updated every time the user changes the selected search.

```
export default createStore({
  state: {
    searches: [],
    tweets: [],
    BarChartList: [],
    LineChartList: [],
    TopPosts: [],
    TopUsers: [],
    activity: {}
  },
},
```

**Listing 4.3:** First version of the Vuex store

The main difference between the previous design choice and the new implementation passes the object to the displayed list which is the *tweets* list. Instead of extracting each attribute in the store and setting the attribute as a variable, the entire object is added to *tweets* list displayed. Inside the dashboard, the attributes are fetched for each respectable component. The implementation listed below makes the code cleaner, and displays the components faster.

```
export default createStore({
  state: {
    //List with objects that contain the title and if the title is active or not
    searches: [],
    //The tweets that is displayed MAX 2 queries
    tweets: [],
    //List that contains all of the data the user has added to the search list
    allTweets: []
  }
},
```

**Listing 4.4:** Final version of the Vuex store

In the dashboard component, the list of data that has been defined in the "store"(the file where the application state is held) is called on with the help of the getters defined in the

"store. The data is retrieved in two different computed functions. We have two computed functions to call on the data from the store because of the opportunity to display two different queries at the same time. Essentially the first computed function contains the first API-request and the second computed function contains the second API-request.

With the two computed functions in the dashboard component, it is now possible to put these as props in the different sub components that the dashboard component consists of. The computed functions are passed as props in the sub components.

Props allows us to pass data from a parent component down to a child component, in this case, the parent component would be the dashboard component and the child component would be all the components that the dashboard component contains.

Once the computed function is passed as a prop to the child components, it is possible to access the data from the child component and thus use this data for visualization. An example would be the line chart component, one of the many child components of the dashboard component. Once the computed function is passed in the dashboard component as a prop in the line chart component, it is possible to use the prop locally in the line chart component and access the prop's data.



# Chapter 5

## Discussion

### 5.1 Problems and challenges

The first and the most obvious problem is the amount of data returned from the Twitter API. Because the Twitter API only returns one hundred tweets from the last seven days, many queries containing fake news circulating on Twitter from earlier than this would not return any data.

Because of these restrictions from the API, the application did not work as a tool for checking fake news that has been circulating on Twitter previously. The application worked fine as a tool if the searched query is recently posted on Twitter, meaning if we had access to the entire archive of tweets, the application would work as intended. The implementation of pagination made the application closer to what the full archive version would have been, but there are still queries that does not return any data. In the "Premium Twitter API" section, we will explain how it is possible to pay to access the entire archive of tweets.

In the very beginning the web site was developed as a Flask application using HTML, CSS and vanilla JavaScript. This application worked fine for extracting data from the Twitter API and displaying a single chart. We quickly found out however, that this application was not very scalable and not reactive at all. Therefore we decided to scrap the first version and make a new version using Vue.Js. We all had some experience using Vue.Js and knew that the website would be a lot better in regards to user friendliness, scalability and reactivity.

After switching to Vue.js, we had to decide which version of Vue that should be used. We decided to use Vue 3 because it is faster and easier to use than the previous versions. The problems with Vue 3 started to occur when trying to use external Vue extensions because most of the user-made extensions were made for the earlier versions. Vue 3 was released on 18 September 2020 [18], and any extension made before this date would not be compatible with our application. This caused many small tasks like adding tooltips, using D3 Vue, and Bootstrap Vue extensions to be more complex because we had to implement our own versions instead of using the already made extension. For example, the word cloud could be made by using an extension, but we had to use a combination between HTML and CSS to make a word cloud from scratch.

We also wanted to extend the application to other social media platforms like Facebook, but the API is not available for everyone to use as the Twitter and Reddit API. The Facebook API websites headlines this message: "Access to the Public Feed API is restricted to a limited set of media publishers and usage requires prior approval by Facebook. You cannot apply to use the API at this time.". Therefore the application is restricted to two social medias. Facebook would have been a good addition to the application due to the large amount of misinformation being spread there.

## 5.2 Experimental Evaluation

### 5.2.1 Sentiment Analysis

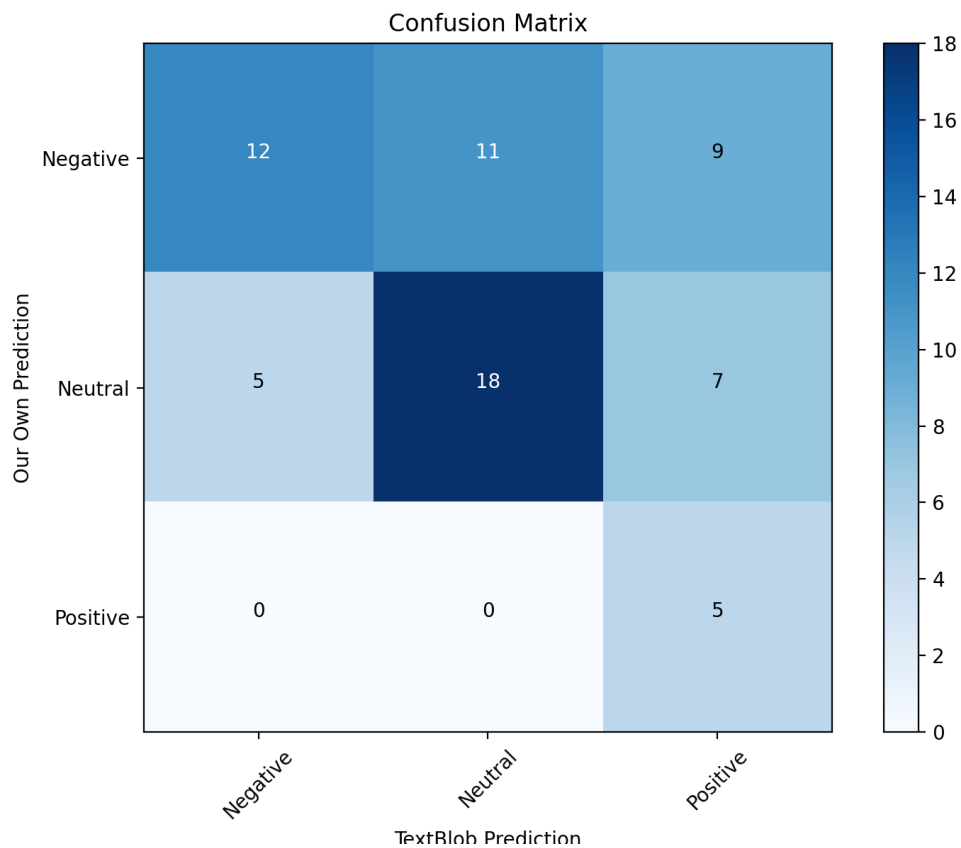
Using TextBlob, a sentiment analysis was done on the searches shown in the table below. In addition to this, we also did our own analysis of the searches to examine the accuracy of TextBlob. The table shows the amount of tweets that TextBlob deem positive, neutral or negative as well as the amount of tweets in a search that we have deemed either positive, negative or neutral.

Testing the accuracy of TextBlob		
Search	TextBlob Results	Our Results
<b>Bill Gates</b>	Positive:21, Neutral:29, Negative:17	Positive:5, Neutral:30, Negative:22
<b>Racism</b>	Positive:40, Neutral:30, Negative:25,	Positive:28, Neutral:25, Negative:42

Only comparing the amount of positive, negative and neutral tweets that TextBlob predicted with our own was a bit thin when controlling the accuracy and "trustworthiness" of the TextBlob sentiments. Therefore we decided to create confusion matrices to further



look at the accuracy of the classifications done by TextBlob. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known, in this case the true values being our own interpretation of a tweets sentiment..



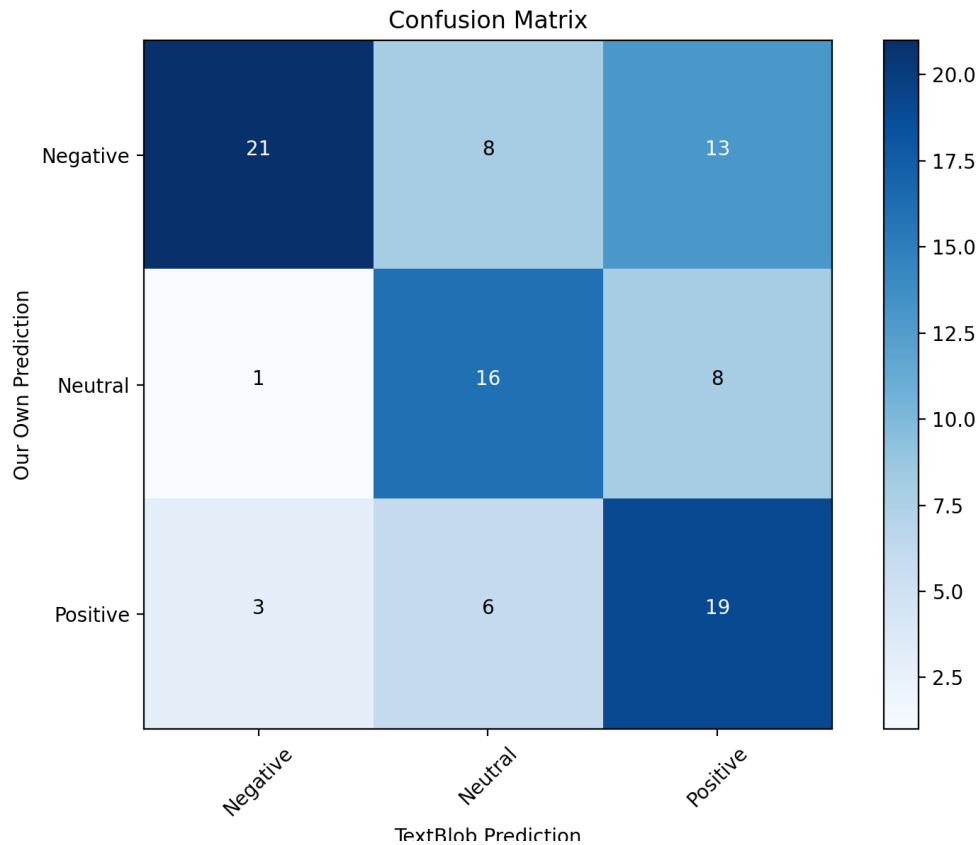
**Figure 5.1:** Confusion Matrix for search "Bill Gates"

For the search Bill Gates we got 67 unique tweets. The confusion matrix shows that TextBlob predicted 17 negative, 29 neutral and 21 positive tweets whilst we deemed 32 negative, 30 neutral and 5 positive tweets.

Of the 17 tweets that TextBlob predicted to be negative tweets, we deemed 12 to be negative as well. Additionally TextBlob predicted 5 other tweets as negative which we deemed to be neutral.

TextBlob also predicted 29 tweets as neutral, of these 29 tweets we deemed 18 as neutral as well. The additional tweets predicted as neutral from TextBlob was actually deemed negative by us.

As for the positive tweets, TextBlob predicted 21 tweets to be positive where as only 5 of these tweets were also deemed positive by us. This tells us that for the positive tweets there was worse correlation between our own interpretation and TextBlobs predictions compared to the negative and neutral tweets.



**Figure 5.2:** Confusion Matrix for search "Racism"

For the search racism there was 95 unique tweets. By looking at the confusion matrix one can see that TextBlob predicted 40 tweets as positive, 30 as neutral and 25 as negative. We on the other hand interpreted 28 tweets as positive, 25 as neutral and 42 as negative.

Of the 25 tweets that TextBlob predicted as negative, we predicted 21 tweets to also be negative. TextBlob also predicted 4 additional tweets as negative where we predicted 3 of these as positive and 1 of them as neutral. Thus we can see that the correlation between our own predictions and TextBlobs predictions is high when it comes to the negative tweets for the search 'Racism'.

With the neutral tweets, TextBlob predicted 25 tweets as neutral, 16 of which we also deemed to be neutral. In addition to this TextBlob also predicted 14 additional tweets as neutral and of these 14 tweets we deemed 6 as positive and 8 as neutral. This tells us that the accuracy of the neutral tweets are somewhat high for this search.

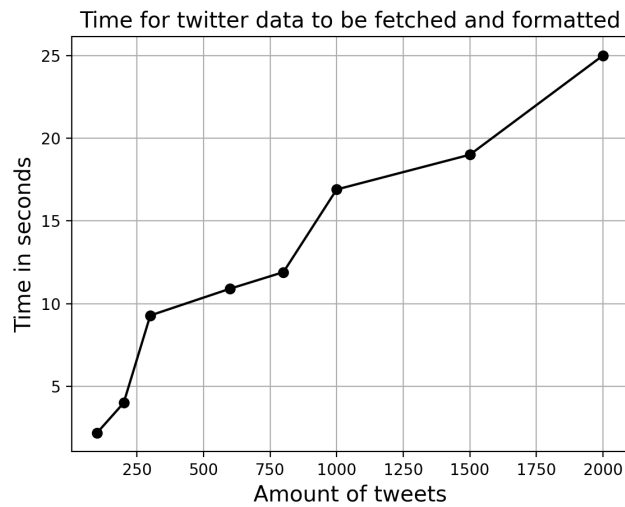
As for the positive tweets, TextBlob predicted 40 of these, of which we predicted 19 of these as positive. TextBlob also predicted 21 additional tweets as positive, and of these 21 we deemed 8 as neutral and 13 as negative. Although there are a lot of same predictions of the positive tweets there are also a lot of tweets that are differently predicted when it

comes to the positive, therefore one can say that the accuracy is somewhat low for the positive tweets in this search.

The analysis that we have done above goes to show that TextBlob is far from one hundred percent accurate even though it is a good pinpointer of the sentiment of a tweet. When examining the tweets ourselves we observed that there were obviously positive tweets that were predicted as negative by TextBlob or vice versa. More often than not, this happens when TextBlob reads a clearly negative word in a positive tweet, for instance a curse word. In other words TextBlob doesn't always understand the context of the words in the tweet and thus gets an incorrect prediction. On the other side, the "controlling" organ in this analysis has been ourselves and even though we tried to be as unbiased as possible we do understand that this is impossible to fully do and thus our interpretation of a tweet is relative to us. A negative tweet in our eyes does not necessarily have to be deemed negative by the next man and so we have had this in mind when analysing the TextBlob sentiments.

### 5.2.2 Performance Evaluation

In order to test the scalability of the application, there must be conducted an experiment to test how long the application uses to process the different amounts of data. The experiment was conducted by timing how long the back-end uses fetch and format the tweets from the raw Twitter data to when the data is ready to be displayed in the front-end. This was done by formatting one hundred to two thousand tweets to check if the application could handle large amounts of data and check if any amounts are causing a spike in run time. Below is a plot of how the run time of the function that is fetching and formatting the tweets:



**Figure 5.3:** Loading time for tweets

By reading the plot, one can see that there is a very minimal difference from a hundred to two hundred. However, there is a sudden spike in the time at three hundred tweets; one can also notice a spike at around a thousand. There is also a steady increase from 300 to 800 and 1000 to 2000 tweets.

Evaluating the results shows the application can handle more significant amounts of data but at the cost of a higher loading time. There is also a conclusion to use a maximum of 500 tweets because this amount is a combination of enough data to display the spread and a reasonable waiting time for the user.

### 5.2.3 Feedback Survey

To get a glimpse of the website from the outside, the website was sent out to a group of ten people to test. In order to get realistic feedback the test group consisted of tutors, computer science students, and people with zero coding experience. After the users had tried out the website, a form was submitted answering questions about the functionality, design, and an open field where the positive and negative feedback could be submitted. Using a feedback survey allows the developers to understand what is done correctly and what could have been improved. Test users may also encounter bugs and errors that have been overlooked.

The test users were asked to rate the website on a scale of 1-5 in five different categories ranging from the overall rating to how functional the website is. The overall rating from the test users averaged a bit higher than four showing the overall impression of the website was positive. The same goes for the design of the website. The question about how functional the website was, the replies were slightly worse. We were familiar with this since some of the searches did not return any data, but the response was better than expected.

The responses for the question: "How well do you think the website displays the spread of a claim?" clearly got the best rating of the questions. Getting positive feedback on this part of the application was great because much focus was used to create as many relevant components for displaying the spread of a claim. Getting positive feedback is great after working hard on a project, but with good news, there is also some bad news. The question where the website scored the worse was about the loading time from adding a search to the data is ready to be displayed. Most test users thought the loading time was a bit too long, and some said it was as expected. After the feedback we did some slight improvements in enhancing the performance, but the waiting time for the API will always be the same therefore the loading time did not improve as much as we wanted it to.

We received feedback from the users that the comparison view was a good addition, a clean look, and they liked the charts. There were some complaints about the loading time for a query, some design issues, and some unknown bugs were discovered. The negative feedback from the test users was constructive and helped us fix some unknown issues; for example, test users using windows would get a scroll bar that should not be there that mac users would not get.

## 5.3 Further Development

An idea for a further developed version of the website is a website where the user can choose different queries to track constantly. For example, the user chooses a query, and from the moment it is added, the API is called with the query every hour, and the returned data would be stored in a database. This implementation would make the application a more useful tool and more comprehensive analysis of the queries in relation to the current application. A database would need to be implemented instead of the locally stored data in the Vuex store, and the back-end would need to be redesigned to call the API every hour. A service like this could be monetized by requiring a monthly fee for the subscription. There is also an opportunity to use machine learning to the sentiment component. The current sentiment analysis uses an external API, but in a further developed version, storing the data and using machine learning to improve the sentiment analysis is possible.

### 5.3.1 Premium Twitter API

When developing this application, we quickly found out that we do not get as much data from the API as we were hoping for, we tried doing multiple API calls, but the API would return more duplicates for each call. So the data shown on the website is nowhere close to the actual data; the particular reason for this is because the free Twitter API only returns one hundred tweets from the last seven days (with some irregularity). However, it is possible to fetch the actual data from the API to create a tool that displays all of the data. To access the actual data, it is required to buy the enterprise version of the API. The enterprise API has two different enterprise search API's: 1. 30-Day Search API provides data from the previous 30 days. 2. Full-Archive Search API provides complete and instant access to the entire corpus of Twitter data dating back to the first Tweet in March 2006.

Having access to the enterprise API, we could create a website that displayed all of the data from the last 30 days or for the whole duration Twitter existed. Accessing all of this data would also create more problems concerning data storage. If this website were to be scaled to display all of the data, we would need to implement a database to store the data.

## 5.4 Conclusion

In this project we created a website for examining the spread of fake news on the two social media platforms Twitter and Reddit. The first goal was to visualize the data returned from the APIs; this was achieved by creating components that displays the total engagement, the top posts and most influential users. The second goal was to visualize the spread of the fake news. This was achieved by creating components such as the node network which shows who has posted a tweet and who has interacted with it. The last goal was to allow the users to compare two queries that have been searched for. We managed to solve this by creating a dual view option in the dashboard. Although we managed to achieve all of the goals that were set for the application, there are still some minor complications revolving fetching data from the APIs. If the query is not somewhat relevant the application does not return any data. As per now the application is deployed and available for anyone to use.





# List of Figures

3.1	Block Diagram	13
3.2	Flowchart on how the different scripts communicate with eachother.	24
3.3	Sequence Diagram	26
4.1	Landing page	28
4.2	Authentication page	29
4.3	Your Trackers	30
4.4	Dashboard page	30
4.5	Tooltip component	32
4.6	Searchlist component	33
4.7	Engagement component	34
4.8	Linechart component	34
4.9	Barchart component	35
4.10	Geochart component	36
4.11	Node network component	37
4.12	Top posts and most influential users components	37
4.13	Wordcloud component	38
4.14	Twitter sentiment component	39
5.1	Confusion Matrix for search "Bill Gates"	45
5.2	Confusion Matrix for search "Racism"	46
5.3	Loading time for tweets	48



# List of Listings

2.1	Using the Chartkick.js library	10
3.1	Default Twitter API response	15
3.2	Twitter API response with parameters	15
3.3	Data returned for a user	16
3.4	URL encoding	16
3.5	Directly accessing the Reddit API	16
3.6	Example of a Reddit instance:	17
3.7	Calling the Reddit API	17
3.8	Calling the Reddit API	17
3.9	Method for fetching Twitter data	18
3.10	Method for fetching Reddit data	18
3.11	Pagination implementation	20
3.12	JSON response for no data	21
3.13	Formatting the data for the bar chart	22
3.14	Formatting the data for the engagement component	23
3.15	How the query is fetched and the data is sent to the front-end	24
3.16	How the data is retrived in the front-end	24
3.17	How the front-end communicates with the back-end	25
4.1	Importing the different components into the dashboard component	31
4.2	Calling the Mapquest API with Geocoder	36
4.3	First version of the Vuex store	40
4.4	Final version of the Vuex store	40



# List of Tables

1.1 Contributions . . . . .	4
-----------------------------	---



# Bibliography

- [1] Hoaxy. <https://hoaxy.osome.iu.edu/>, Retrieved March 2021.
- [2] Social listening definition. <https://www.mycustomer.com/hr-glossary/social-listening>, Retrieved April 2021.
- [3] Github front-end. <https://github.com/RamtinHaf/FNT-FRONTEND>, Retrieved May 2021.
- [4] Github back-end. <https://github.com/RamtinHaf/FNT-BACKEND-b>, Retrieved May 2021.
- [5] BuzzFeed. <https://abcnews.go.com/Technology/fake-news-stories-make-real-news-headlines/story?id=43845383>, Retrieved April 2021.
- [6] Mit. <https://science.sciencemag.org/content/359/6380/1146>, Retrieved April 2021.
- [7] Html: Hypertext markup language. <https://developer.mozilla.org/en-US/docs/Web/HTML>, Retrieved March 2021.
- [8] Css. <https://en.wikipedia.org/wiki/CSS>, Retrieved March 2021.
- [9] D3. <https://d3js.org/>, Retrieved April 2021.
- [10] Twitter: Twitter wikipedia. <https://en.wikipedia.org/wiki/Twitter>, Retrieved April 2021.
- [11] Twitterstats. <https://www.dsayce.com/social-media/tweets-day/>, Retrieved April 2021.
- [12] Reddit: Wikipedia. <https://no.wikipedia.org/wiki/Reddit>, Retrieved April 2021.
- [13] Praw: Python reddit api wrapper. <https://github.com/praw-dev/praw>, Retrieved April 2021.
- [14] Twitter. <https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent>, Retrieved May 2021.

- [15] Twitter. <https://developer.twitter.com/en/docs/twitter-api/rate-limits>, Retrieved May 2021.
- [16] Geocoder. <https://geocoder.readthedocs.io/>, Retrieved May 2021.
- [17] Mapquest. <https://developer.mapquest.com/documentation/geocoding-api/>, Retrieved May 2021.
- [18] Vue3: Release date. <https://madewithvuejs.com/blog/vue-3-roundup>, Retrieved April 2021.