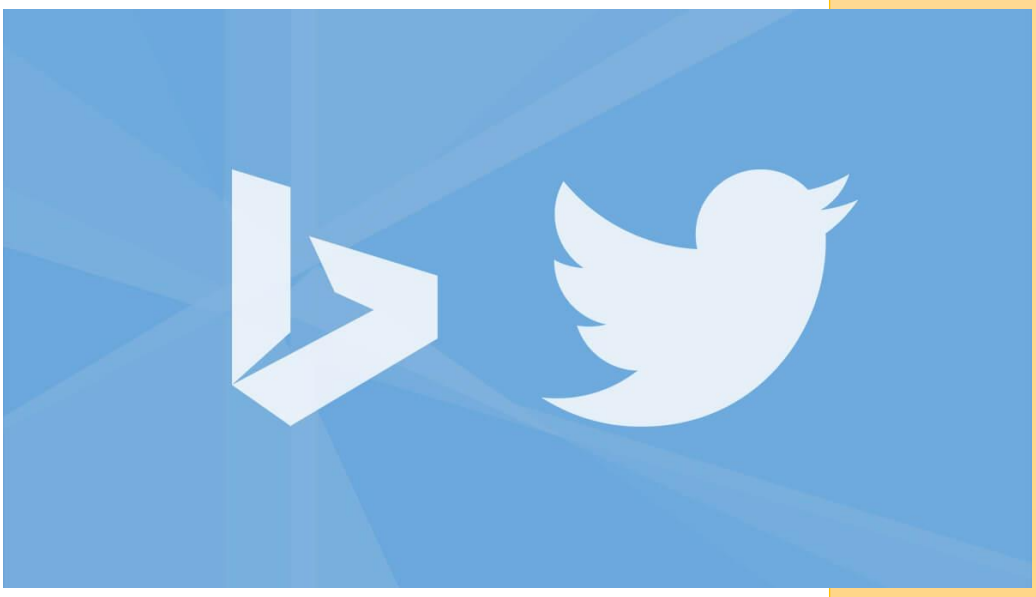
2020

Trend/Bing



CAB432 Assignment 1

Mark Burton

N9801154

9/9/2020

Contents

Introduction

Mashup Purpose & description

The purpose of this mash up is to serve up news articles and images related to top trending topics for specific locations. The user is able to enter the name of a town, city, state or country and the app will communicate with the Twitter API, and direct the user to a page listing the top 10 trending topics for that location. The user is then able to select either news or images relating to each trend and the app communicates with either the Bing News Search API or the Bing Image Search API, respectively, to retrieve and display the requested news or image data to the user.

This mashup serves the purpose of providing news articles and images from a variety of sources to give users more information on trending topics for locations across the world. Social media has a reputation for providing unreliable information, however, Twitter is able to provide credible data regarding the trending topics of its users for certain locations. Combining these trends with news articles from more credible sources, provided by news aggregator Bing news, offers users far better information on trending topics than what may be provided in associated tweets. Similarly, the app's ability to provide a selection of images, from a variety of sources, relative to a trend gives the user an unbiased overview of trend topics that would not be available on Twitter.

The mashup app has been designed to appear clean and uncluttered whilst still providing users with a smooth in retrieving informative information for any location quickly.

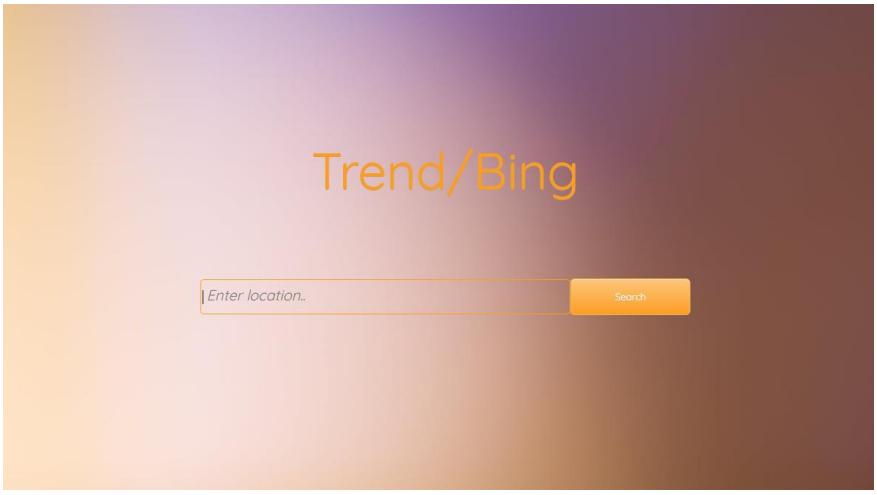


Figure 1 - The main page of the app appears clean and uncluttered.

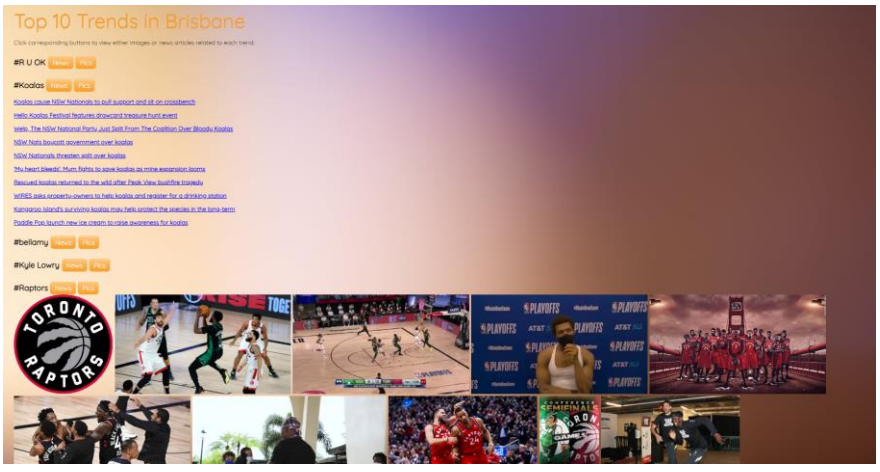


Figure 2 - Users are able to retrieve informative information quickly for any location.

Services used

The mashup app uses the Twitter API, the Bing News Search API and the Bing Image Search API. Three separate endpoints of the Twitter API are required along with one endpoint for each of the Bing APIs, for a total of five endpoints. The three twitter APIs are used sequentially to return the trend data of the location closest to the user entered location that Twitter is able to provide trend data for. Each of the end points that are used in the app are detailed below.

Twitter Geo Search API (v.1.1)

Returns a list of valid locations matching the queried place name that are recognised by Twitter. An example response of this endpoint can be seen in Appendix A.

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

Docs: https://developer.twitter.com/en/docs/twitter-api/v1/geo/places-near-location/api-reference/get-geo-search

Twitter Trends Closest API (v.1.1)

Returns the locations that Twitter has trending topic information for, closest to a specified location. An example response of this endpoint can be seen in Appendix B.

Endpoint: `https://api.twitter.com/1.1/trends/closest.json`

Docs: https://developer.twitter.com/en/docs/twitter-api/v1/trends/locations-with-trending-topics/api-reference/get-trends-closest

Twitter Trends Place API (v.1.1)

Returns the top 50 trending topics for a specific `WOEID`, if trending information is available for it. An example response of this endpoint can be seen in Appendix C.

Endpoint: `https://api.twitter.com/1.1/trends/place.json`

Docs: https://developer.twitter.com/en/docs/twitter-api/v1/trends/trends-for-location/api-reference/get-trends-place

Bing News Search API (v7)

Returns a list of relevant news articles matching a specified query.

Endpoint: `https://api.cognitive.microsoft.com/bing/v7.0/news/search`

Docs: https://docs.microsoft.com/en-us/rest/api/cognitiveservices-bingsearch/bing-news-api-v7-reference

Bing Image Search API (v7)

Returns a collection of relevant images matching a specified query.

Endpoint: `https://api.cognitive.microsoft.com/bing/v7.0/images/search`

Docs: https://docs.microsoft.com/en-us/rest/api/cognitiveservices-bingsearch/bing-images-api-v7-reference

Mashup Use Cases and Services

To better illustrate the services that this mashup app provides, the following use cases demonstrate how it may be used by different people.

News Articles for Local Trends

As an	Local resident
I want	To be able to read news articles of popular trends in my area
So that	I can keep up with current events relative to my community

In order to achieve this use case, the resident enters their home town into the text input field of the index page. The app then uses the Twitter API to retrieve the top trending topics for that area and redirects the resident to the "trending" route which displays the top 10 trends for the resident's home town. The resident would then press the "News" button of the trends that they wish to view news articles for. When a "News" button is pressed the app contacts the Bing News Search API and retrieves the top 10 news articles for that trend. It then updates the page to display the titles of the news articles which also serve as URL links to the news article pages themselves.

Image Viewer for Destination Trends

As a	Frequent traveler
I want	To be able to view images of trending topics for my next destination
So that	I can see what's currently hot at that location

In order to achieve this use case, the traveler enters their travel destination into the text input field of the index page. The app then uses the Twitter API to retrieve the top trending topics for the location and redirects the traveler to the "trending" route which displays the top 10 trends for the traveler's next destination. The traveler would then press the "Pics" button of the trends that they wish to view images for. When a "Pics" button is pressed the app contacts the Bing Image Search API and retrieves the top 20 Images for that trend. It then updates the page to display a collage of the images to the traveler.

Insight to Community Interests

As a	Traveling salesperson
I want	To get a quick overview of what's popular at each of the towns that I visit
So that	I can tailor my sales pitches to suite the current interest of my audience

In order to achieve this use case, the salesperson enters a town that they will visit into the text input field of the index page. The app then uses the Twitter API to retrieve the top trending topics for that town and redirects the salesperson to the "trending" route which displays the top 10 trends for the town. If the salesperson would like to view images for a particular trend then they may press the "Pics" button. The app contacts the Bing Image Search API and retrieves the top 20 Images for that trend. It then updates the page to display a collage of the images to the traveler. If the salesperson is intrigued by the images of a trend and wishes to view news articles for the same trend then they could press the "News" button. When the "News" button is pressed the app contacts the Bing News Search API and retrieves the top 10 news articles for that trend. It then updates the page to hide the currently displayed images and instead display the titles of the news articles which also serve as URL links to the news article pages themselves.

Overview of World Trends

As a	US President
I want	To view *and understand* what is trending on Twitter worldwide
So that	I can tweet about these trends myself and pretend I know what I am talking about

In order to achieve this use case, the president leaves the text input field of the index page blank. The app then uses the Twitter API to retrieve the top trending topics worldwide and redirects the president to the "trending" route which displays the top 10 trends in the world. If the president would like to view news articles for a particular trend then he may press the "News" button. The app then contacts the Bing News Search API and retrieves the top 10 news articles for that trend. It then updates the page to display the titles of the news articles which also serve as URL links to the news article pages themselves. If the president then decides that he prefers to look at pictures rather then read news articles then he could press the "Pics" button. When the "Pics" button is pressed the app contacts the Bing Image Search API and retrieves the top 20 Images for that trend. It then updates the page to hide the currently displayed news articles and instead display a collage of images.

Technical breakdown

When a new connection is made to the app it is directed to the appropriate route based upon the URL. The index route, is the home page of the app and is designed to have a clean and simple appearance. This page features a main heading and an item that comprises of an input text field and a button. The user is prompted to enter a location into the input text field by a subtle placeholder. Pressing either the 'search' button or the 'enter' keyboard key will take the text entered in the input field as the search term to redirect the user to the trending route.

If a search query is received then the 'trending' route will make three separate calls to the Twitter API. The first request is made to the geo/search endpoint and uses the location entered by the user as the query parameter. This endpoint returns the valid locations recognized by twitter that match the queried location name. The coordinates of the top resulting location are used in the next sequential request to the trends/closest endpoint. This endpoint returns the closest location to the coordinates that Twitter is able to return trending topics for. The WOEID of the top resulting location is then used for the third request to the trends/place endpoint. This endpoint returns the top 50 trends for the WOEID location, of which, the top 10 are used to render the 'trending' page.
In the case that there is no search query for the trending route, then the first two endpoint requests are skipped and only the trends/place endpoint will be used. A WOEID of 1, which Twitter recognizes as worldwide, is used in this instance causing the endpoint to return the top trends worldwide. Again, the top 10 of these trends are used to render the 'trending' page.

The 'trending' page features; a heading which specifies the location of the top 10 trends along with a brief explanation of how to use the app. This page then displays 10 items containing the top 10 trends for the location along with 'News' and 'Pics' buttons. When one of the buttons is pressed the app displays the requested data returned from either the bingNews or bingImages routes. In the case of the 'News' button, the bingNews route uses the Bing News Search API to retrieve the top 10 news articles related to that trend and sends it to the client. Likewise, the bingImages route uses the Bing Images Search API to retrieve the top 20 images related to that trend and sends it to the client. Once the data has been received on the client-side, it is displayed to the user beneath the trend that it is associated with. In order to keep this page uncluttered, it is not possible to display both news stories and images for the same trend at the same time. If news stories are currently been displayed and the Pics button is pressed then the news stories will be removed before the requested images are displayed to the user. Similarly, if the news button is pressed whilst images are currently displayed for the same trend then the images will be removed before the requested news articles are displayed.

Finally, if a button Is pressed whilst it's corresponding data is already been displayed then that data will simply be removed and no fetch request will be made to the server.

This mashup is built using the express application framework and utilizes pug engine support for generating the front-end HTML. The first step of building the application was to generate an empty express app using express-generator. Then, the server-side of the app was developed starting with implementation of the communication with the required API endpoints for Twitter and Bing. The first Twitter API endpoint, geo/search, was reached first using NPM package 'twit', then, subsequently the other Twitter endpoints, trends/closest and trends/place were reached in the same way. The response data of each of these endpoints was then manipulated into the request parameters of the other endpoints until the correct trends for a given location could be received. Next, both the Bing News Search API and Bing Image Search API were reached using the NPM package 'node-bing-api'. Once all of the required API endpoints could be reached and the required data retrieved from each, they were used in the implementation of the app routes; twitter, bingImages and bingNews. To complete the server-side JavaScript component of the app an index route was built to serve the home page of the web app.

The next stage of the building process of the app was to start building the HTML component to be send to the client-side of the app and displayed to the user. For this the pug engine was used to generate the HTML documents, starting with the 'layout' view which defines the title and favicon along with a background, all inherited by the 'index' and 'trending' user views. The 'trending' view was then created featuring a main heading informing of the location that the displayed trends belong to, a brief explanation of how to view news or images related to the trends, and the trends themselves along with 'News' and 'Pics' buttons for each. Once the 'trending' page was displaying correctly, a client-side JavaScript was built to handle the button click events. The client-side JavaScript retrieves data from either the bingImages or bingNews routes depending on which button is pressed and updates the 'trending' page to display the required data to the user. Alternatively, if the data relating to the button that is pressed is already showing then it will be removed and, if data of a different type will be replaced with the most recently requested data type. The 'index' view was then created featuring a main heading and an input text box and search button to receive the location that the user wants to receive trending data for. A client-side JavaScript was attached to the 'index' view that would serve to retrieve the user input text from the text box and use it to redirect the user to the 'trending' page.

When implementing the twitter route, issues were encountered when sequentially making requests to the three Twitter endpoints. The issue was caused by stringing callback functions together, resulting in complex and confusing code commonly referred to as known as callback hell. In order to resolve this issue promises were utilized to make the code less confusing and easier to work with. Luckily, the 'twit' package get function supports the return of promises which are objects that represent the eventual completion (or failure) of an asynchronous operation and its resulting value.

Another challenge that was overcome as part of the implementation of this mashup web application was an inexperience with HTML, CSS and PUG which made building the front end of the application difficult. This was overcome by reading lots of online material relative to the front-end development of web apps complimented by you tube videos. The result is an elegantly simple implementation that is able to achieve its purpose whilst remaining a clean, uncluttered appearance.

Architecture and Data Flow

NPM package Express-Generator was used to generate a template for the application. Additionally, Pug engine support was added to the app generation to assist in the development of the app's frontend. Part of the express-generated code, 'bin/www', sets out the network parameters of the app. It sets the port number that the app runs on to 3000 and uses it to create a http server.

When a new connection is made to the app, app.js receives the connection and either routes it to the appropriate route based upon the URL, or, handles any errors. The 4 routes that are defined in app.js are:

- '/' – This is the route to the user interface index page of the app and directs the app to the index router, 'routes/index.js'.
- '/trending?' - This is the route to the user interface trending page of the app and directs the app to the trending router, 'routes/twitter.js'.
- '/trending/news?' - This is the route used by the client side of the 'trending' page which fetches news articles from the Bing news router 'routes/bingNews.js'.
- '/trending/pics?' - This is the route used by the client side of the 'trending' page which fetches images from the Bing image router 'routes/bingImages.js'.

The index route, handled in the 'routes/index.js' file, is the home page of the app and is rendered using the 'index.pug' file, shown in figure 3. This page features; a main heading (h1), and an item that comprises of an input text field and a button. Pressing either the 'search' button or the 'enter' keyboard key will trigger a client-side action listener which takes the text entered in the input field as the search term to redirect the user to the trending route.

views > index.pug

```
1  extends layout
2
3  block content
4    h1.index Trend/Bing
5    .item(class='index')
6      input(type='text' name='q' placeholder=' Enter location..').input
7      button(id='Search' class='index').search Search
8
9      script(src='/javascripts/index.js')
10
```

Figure 3 - PUG engine file for the index.

The trending route, handled in the 'routes/twitter.js' file, has two router.get functions, one is used to process URL requests containing a search query and another is used to handle empty requests that do not contain a search query.

If a search query is received then this route will sequentially make three separate calls to the Twitter API, as shown in figure 5. The first request is made to the geo/search endpoint and uses a location as the query parameter. The coordinates of the top resulting location are then extracted from the response and the latitude and longitude values are used as the parameters to the next request in the sequence, the trends/closest endpoint. The default scale of the locations that Twitter is able to provide trending topic data for is 'neighborhood'. The WOEID of the top resulting location is then used as the id parameter to the next request in the sequence, the trends/place endpoint. Additionally, 'hashtags' is included as the exclude parameter to exclude any hashtags from the result trends. This endpoint then returns the top 50 trends for the WOEID location, of which, the top 10 are extracted and added to a string array using a for loop. Once the top 10 trends for the entered location have

views > trending.pug

```
1  extends layout
2
3  block content
4    h1.trending Top 10 Trends in #{location}
5    p Click corresponding buttons to view either images or news articles related to each trend.
6    each trend in trending
7      .item
8        h2 ##{trend}
9          button.news News
10         |
11         button.pics Pics
12
13   script(src='/javascripts/trending.js')
14
```

Figure 4 - PUG engine file for the trending page.

been retrieved from the Twitter API, they are passed to the 'trending' page which is rendered using the 'trending.pug' file, shown in figure 4.
In the case that there is no search query for the trending route, then the first two endpoint requests are skipped and only the trends/place endpoint will be used. A default WOEID of 1, which Twitter recognizes as worldwide, is used in this instance causing the endpoint to return the top trends worldwide. Again, the top 10 of these trends are extracted into and array which is then used to render the 'trending' page using the 'trending.pug' file.

```
// Trends route handler
router.get('/:query', (req, res) => {
  let params = { query: req.params.query }
  T.get('geo/search', params) // Query Twitter API to get coordinates for the entered location

    .then(result => {
      let params = { // Set params
        lat: result.data.result.places[0].centroid[1], // Extract latitude from result data
        long: result.data.result.places[0].centroid[0] // Extract longditude from result data
      }
      return T.get('trends/closest', params) // Query Twitter API to get closest location that trending data is available for

    }).then(result => {
      let params = { // Set params
        id: result.data[0].woeid, // Extract woeid from result data
        exclude: 'hashtags'
      }
      return T.get('trends/place', params) // Query Twitter API to get the top trends closest to the entered location

    }).then(result => {
      const top10Trends = []
      for (let i = 0; i < 10; i++) {
        top10Trends.push(result.data[0].trends[i].name) // Extract the top 10 of the returned trends
      }
      return top10Trends; // return the top 10 trends

    }).then(result => {
      res.render("trending", {  // Render the trending page
        trending: result,
        title: 'Trending in ' + req.params.query,
        location: req.params.query
      });
```

Figure 5 - Part of the 'views/twitter' route showing the three sequential requests made to the Twitter API to return the top 10 trends for a user specified location.

The 'trending' page uses; a h1 element to specify the location that the app is showing the top 10 trends, and a p element which contains a brief explanation of how to use the app. This page then displays 10 items containing the top 10 trends for the location as an h2 element, along with 'News' and 'Pics' buttons. Embedded in this page is also a client side trending.js file, which is used to handle button press events and update the page accordingly. When one of the buttons is pressed an event is triggered in the client-side JavaScript which uses the button type along with the trend of which it is associated with to make a fetch request to either the bingNews or bingImages routes. In the case of the 'News' button, the bingNews route uses the Bing News Search API to retrieve the top 10 news articles related to that trend and sends it to the client-side JavaScript. Likewise, the bingImages route uses the Bing Images Search API to retrieve the top 20 images related to that trend and sends it to the client-side JavaScript. Once the data has been received on the client-side, it is displayed to the user as part of the item of that trend. In order to keep this page uncluttered, it is not possible to display both news stories and images for the same trend at the same time. If news stories are currently been displayed and the Pics button is pressed then the news stories will be removed before the requested images are displayed to the user. Similarly, if the news button is pressed whilst images are currently displayed for the same trend then the images will be removed before the requested news articles are displayed. Finally, if a button Is pressed whilst it's corresponding data is already been displayed then that data will simply be removed and no fetch request will be made to the server.

The routes of the application, found in the 'routes' directory, are all handled on the server. These contain all the code required to interface with the API endpoints that are used. Additionally, all processing of the data that is received from the API endpoints is done in these files in the client-side. Also, on the server side are the PUG engine files, found in the 'views' directory, that are used to generate the HTML that is sent to the client. These receive data and are rendered from within the server route JavaScript files. The 'public' directory contains client-side data, including the favicon icon, the client-side JavaScript files and the CSS stylesheet. The client-side JavaScript files mostly handle client events such as button presses, however, 'public/javascripts/trending.js', is responsible for determining what data is currently being displayed in order to make the correct action based upon which button was pressed. One such action that may be required is to make fetch requests to either the bingNews or bingImage routes and add the corresponding data to the existing HTML. Additionally, this script may be required to remove existing data, display new data or both.

Figure 6 shows a diagram of the data flow of the application.

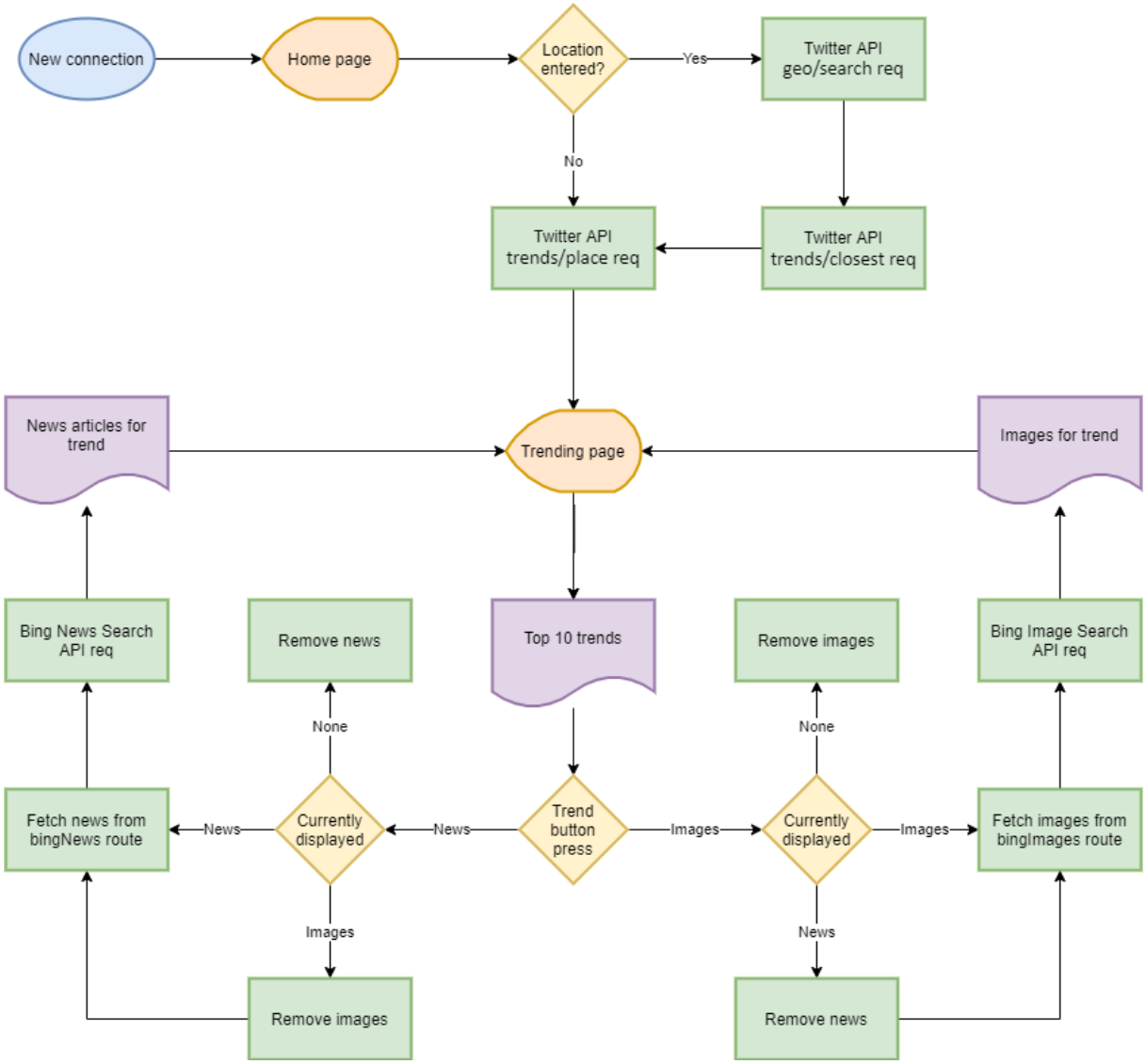


Figure 6 – Flow chart diagram of the data flow of the application

Deployment and the Use of Docker

Docker uses a set of instructions, contained in a text document known as a Dockerfile, to build Docker images. The instructions contained in a Dockerfile can include any command that a user is able to enter into the command line to manually build a Docker image [1]. Since this web application is required to be deployable on cloud machines, a basic Dockerfile was used to build the image. The instructions contained in the Dockerfile can be seen in Appendix D and set the Node version to the v10 release codenamed Dubnium. This version of Node was chosen as it is the current maintenance LTS version and is an up-to-date and properly maintained major version capable of hosting the web server [2]. The command 'npm install' is run in order to install all of the app dependencies and port '3000' is exposed which enables outside connections to access the app. Other instructions in the Dockerfile copy the app source to '/src' and also set it as the working directory. The final command is used to start the server using the 'npm start' command.

Test plan

In order to test that the final release of the web app meets its design specifications a series of manual tests were carried out. The tests applied to the app along with their expected outcomes and results can be viewed in figure 7 and screenshots of the tests are included in the Appendix.

#	Test	Expected Outcome	Actual Outcome	Pass/Fail	Screenshot/s
1	The web app can be connected to when running on a local network	The web app home page is displayed at http://localhost:3000	The home page is displayed at http://localhost:3000	Pass	Appendix E
2	Entering a location	The web app should redirect to the 'trending' page and display top 10 trends for the entered location	User redirected to the 'trending' page and top 10 trends are displayed for the entered location	Pass	Appendix F
3	Leaving the input text field blank	The web app should redirect to the 'trending' page and display top 10 trends worldwide	User redirected to the 'trending' page and top 10 trends are displayed for the entered location	Pass	Appendix G
4	Pressing 'Pics' button for a trend with no other data showing	Images are displayed for the trend	Images are displayed for the trend	Pass	Appendix H
5	Pressing 'Pics' button for a trend with 'Pics' data	Currently displayed images are removed	Currently displayed images are removed	Pass	Appendix I
6	Pressing 'Pics' button for a trend with 'News' data showing	Currently displayed news articles are removed and replaced with images for the trend	Currently displayed news articles are removed and replaced with images for the trend	Pass	Appendix J
7	Pressing 'News' button for a trend with no other data showing	News articles are displayed for the trend	News articles are displayed for the trend	Pass	Appendix K
8	Pressing 'News' button for a trend with 'News' data showing	Currently displayed News articles are removed	Currently displayed News articles are removed	Pass	Appendix L
9	Pressing 'News' button for a trend with 'Pics' data showing	Currently displayed images are removed and replaced with news articles for the trend	Currently displayed images are removed and replaced with news articles for the trend	Pass	Appendix M
10	Images displayed for multiple trends	Images for multiple trends can be displayed at once	Images for multiple trends can be displayed at once	Pass	Appendix N
11	News displayed for multiple trends	News articles for multiple trends can be displayed at once	News articles for multiple trends can be displayed at once	Pass	Appendix O
12	News and pics displayed for multiple trends	A mix of news articles and images can be displayed for different trends at once	A mix of news articles and images can be displayed for different trends at once	Pass	Appendix P
13	Build Docker image	A Docker image can be built successfully using the Dockerfile	A Docker image can be built successfully using the Dockerfile	Pass	Appendix Q
14	Run Docker image	The Docker image can be succefully deployed in a Docker conatiner and the web app can be accessed	The Docker image can be succefully deployed in a Docker conatiner and the web app can be	Pass	Appendix R

Figure 7 - Table of tests carried out on the app

Difficulties and Exclusions

Some difficulties were experienced when implementing the three sequential requests to the aforementioned Twitter API endpoints required when a location is entered. The problems were caused by staggering callback functions within one another. This caused messy and difficult to understand code that proved very difficult to work with, an issue commonly referred to as 'callback hell'. A solution was sought and a solution found in promises. promises are an object that represents the eventual value returned by an asynchronous function and were introduced into NodeJS in 2011 [3]. Implementing promises, which are supported in the 'twit' NPM package used, vastly improved the appearance of the code and made it much easier to work with.

Another difficulty experienced as part of the development of this mashup was the implementation of the front-end HTML that is displayed in the web browser to the user. This difficulty was due to a lack of experience working with this technology. This difficulty was overcome by learning some basic PUG engine commands to easily generate the required HTML. The main source of this learning came from an informative video provided by Michael Esteban [4]. Applying this method of generating HTML streamlined the front-end development of the mashup and provided a good platform to build upon.

A deviation from the initial design proposal of the mashup application was the decision to drop the use of the Flickr API in favor of the Bing Image Search API. The Flickr API was selected to be the source of images to be displayed in the web application. Whilst working with the Bing News Search API, however, the Bing Image Search API was discovered to provide a more precise image search service and deemed a more effective API for use in this mashup. Additionally, both the Bing News Search API and Bing Image Search API can be used using the same Azure Cognitive Services API account making the web application easier to build.

Extensions

In addition to the Bing News Search API and Bing Image Search API there are other APIs available to Azure Cognitive Services API accounts such as Bing Video Search API which returns links to videos. As an extension to this project these additional services could be added to the functionality. This was not included in this release of the mashup, however, as it did not fall within the design scope of the project.

User guide

When a connection is made to the web application, the user is presented with the home page shown in figure 8.

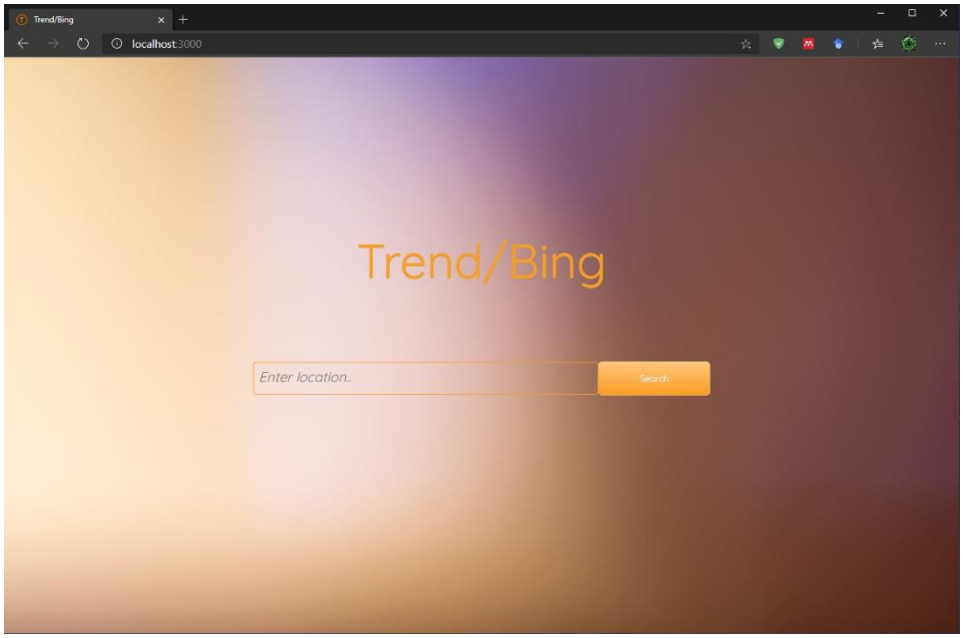


Figure 8 - Home page of the web application

The user is able to enter the name of a location that they wish to find information of the trending topics for. Alternatively, if the text input is left blank then the application will default to find to top worldwide trends.

Either the search button or the enter key on the keyboard will prompt the application to redirect the user to the 'trending' page, as shown in figures 9 & 10.

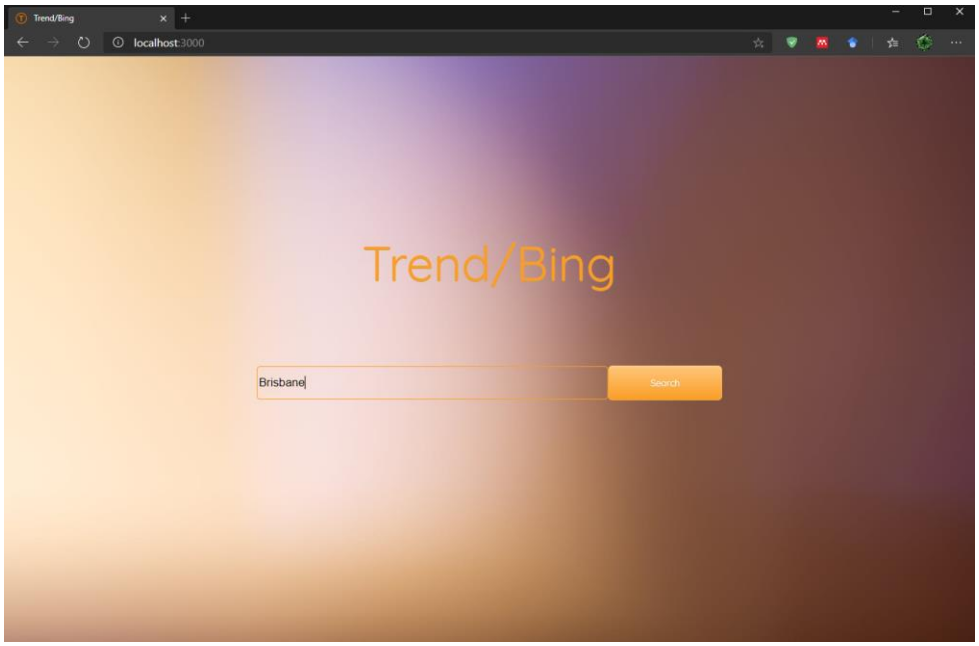


Figure 9 - Home page with a location inputted as a query

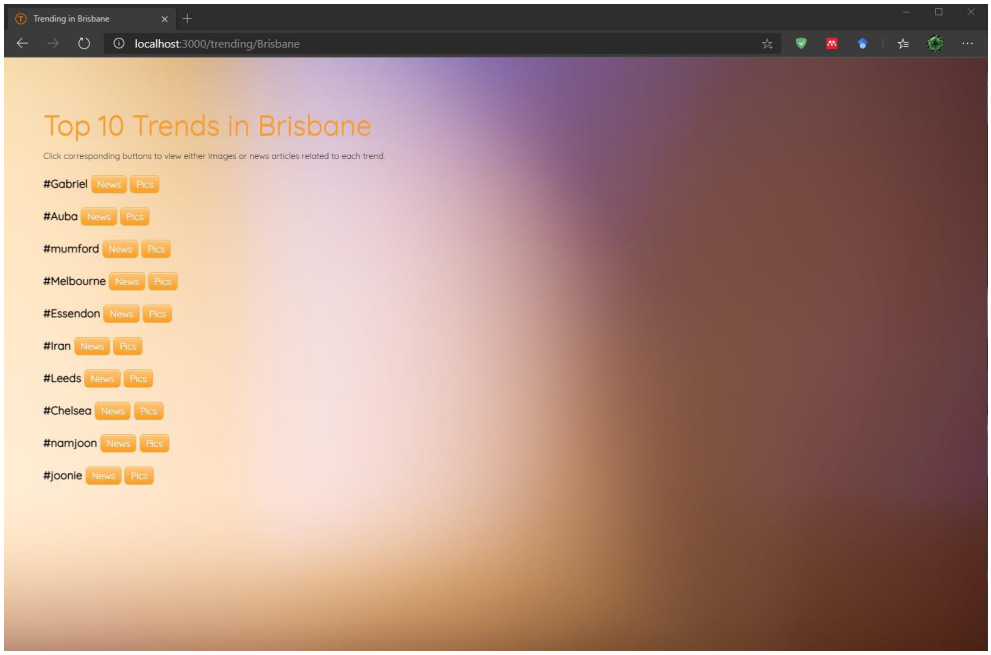


Figure 10 - Trending page displaying the top 10 trends for Brisbane

From this page the user is free to select to view either news articles or images related to each trend using the buttons beside it. Figures 11, 12 and 13 show some examples of news and images being displayed for different trends.

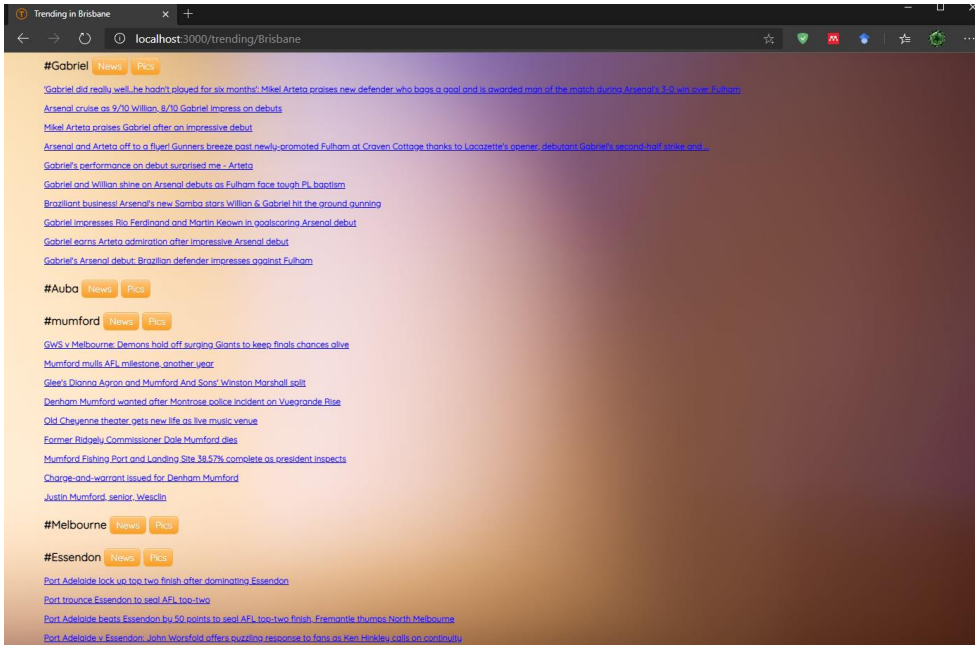


Figure 11 - News stories displayed for different trends

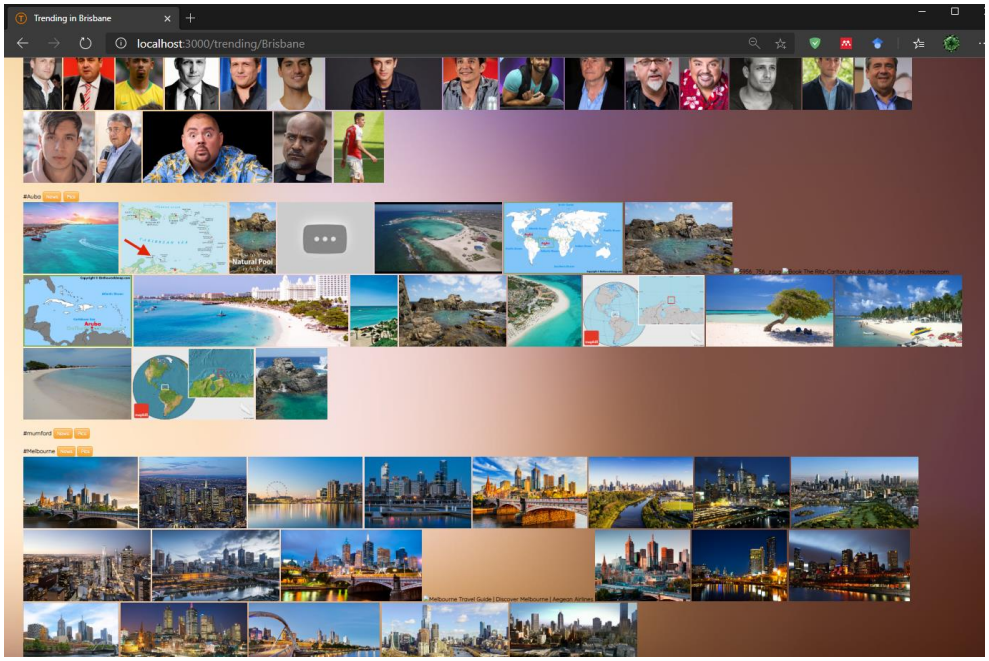


Figure 12 - Images displayed for different trends

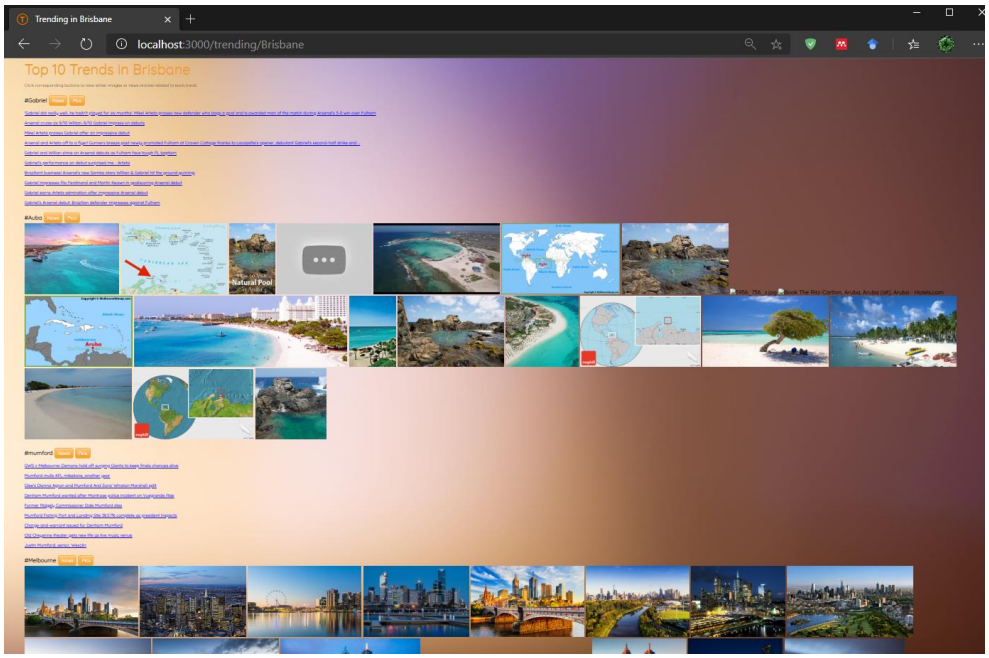


Figure 13 - News articles and images displayed for different trends

The user is able to press buttons more than once and the data displayed will reflect that of the most recent button pressed, unless the data is removed by the most recent button press.

Note: to avoid overly cluttered pages, users are only able to display one of either news articles of images for each trend.

Statement on Assignment Demo

I intend to demonstrate this assignment in a face-to-face format.

References

[1] Docker Inc., “Dockerfile reference | Docker Documentation,” 2020. https://docs.docker.com/engine/reference/builder/ (accessed Sep. 12, 2020).

[2] Node.js “Releases | Node.js.” https://nodejs.org/en/about/releases/ (accessed Sep. 12, 2020).

[3] B. Diuguid “Asynchronous Adventures in JavaScript: Promises | by Benjamin Diuguid | DailyJS | Medium.” https://medium.com/dailyjs/asynchronous-adventures-in-javascript-promises-1e0da27a3b4 (accessed Sep. 13, 2020).

[4] M. Esteban, “*Wikipedia-Overview-Video.mp4” 2020*. https://blackboard.qut.edu.au/bbcswebdav/pid-8822856-dt-content-rid-33907933_1/courses/CAB432_20se2/Wikipedia-Overview-Video.mp4 (accessed Sep. 13, 2020).

Appendix A – Example response of the Twitter API geo/search endpoint


```
{
   "result": {
       "places": [{
           "id": "3797791ff9c0e4c6",
           "url": "https://api.twitter.com/1.1/geo/id/3797791ff9c0e4c6.json",
           "place_type": "city",
           "name": "Toronto",
           "full_name": "Toronto, Ontario",
           "country_code": "CA",
           "country": "Canada",
           "centroid": [-79.27828265214646, 43.629311],
           "bounding_box": {
               "type": "Polygon",
               "coordinates": [
                   [
                       [-79.639319, 43.403221],
                       [-79.639319, 43.855401],
                       [-78.90582, 43.855401],
                       [-78.90582, 43.403221],
                       [-79.639319, 43.403221]
                   ]
               ]
           }
       },
```

Appendix B – Example response of the Twitter API trends/closest endpoint


```
[
  {
    "country": "Australia",
    "countryCode": "AU",
    "name": "Australia",
    "parentid": 1,
    "placeType": {
      "code": 12,
      "name": "Country"
    },
    "url": "http://where.yahooapis.com/v1/place/23424748",
    "woeid": 23424748
  }
]
```

Appendix C – Example response of the Twitter API trends/place endpoint

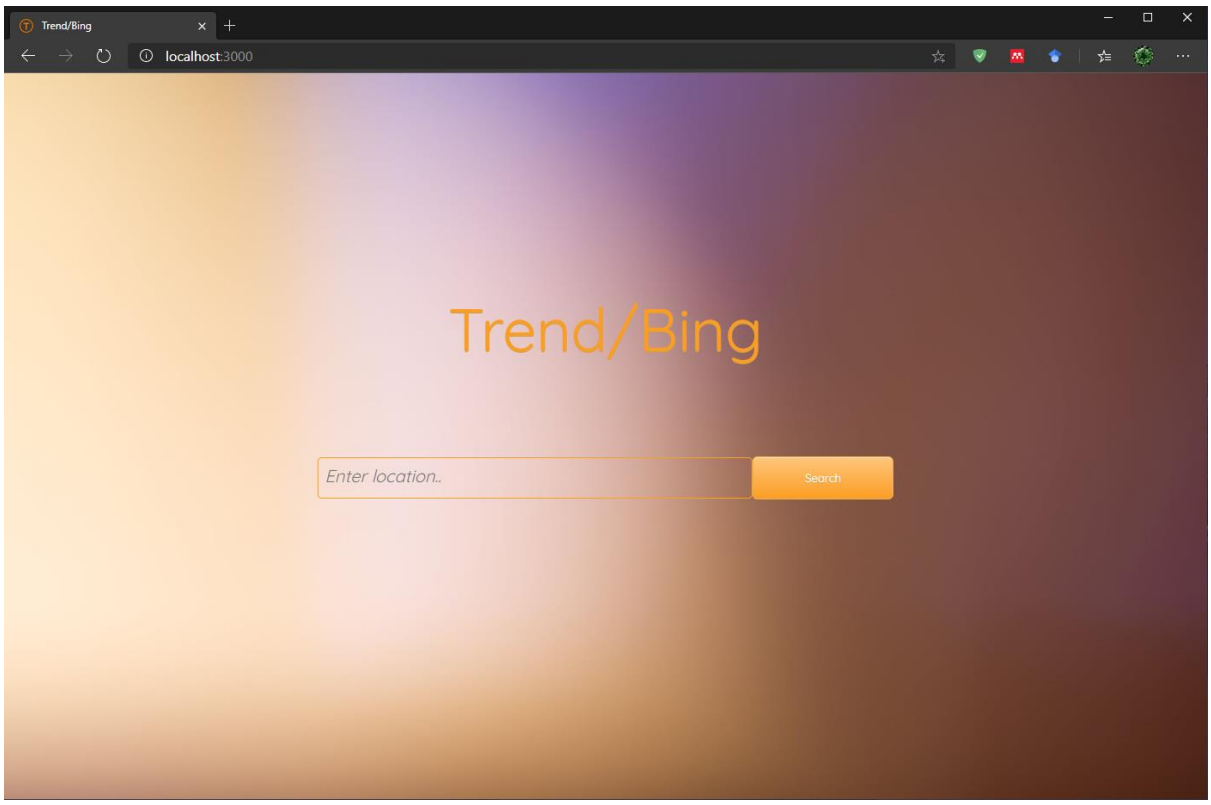

```
[
  {
    "trends": [
      {
        "name": "#ChainedToTheRhythm",
        "url": "http://twitter.com/search?q=%23ChainedToTheRhythm",
        "promoted_content": null,
        "query": "%23ChainedToTheRhythm",
        "tweet_volume": 48857
      },
      {
        "name": "#اليوم_العالمي_للعنبان",
        "url": "http://twitter.com/search?
q=%23%D8%A7%D9%84%D9%8A%D9%88%D9%85_%D8%A7%D9%84%D8%B9%D8%A7%D9%84%D9%85%D9%8A_%D9%84%D9%8
4%D8%B9%D8%AA%D8%A8%D8%A7%D9%86",
        "promoted_content": null,
        "query":
"%23%D8%A7%D9%84%D9%8A%D9%88%D9%85_%D8%A7%D9%84%D8%B9%D8%A7%D9%84%D9%85%D9%8A_%D9%84%D9%84
%D8%B9%D8%AA%D8%A8%D8%A7%D9%86",
        "tweet_volume": null
      },
      {
        "name": "George Lopez",
        "url": "http://twitter.com/search?q=%22George+Lopez%22",
        "promoted_content": null,
        "query": "%22George+Lopez%22",
        "tweet_volume": 90590
      },
```

Appendix D – Dockerfile used to build the Docker image

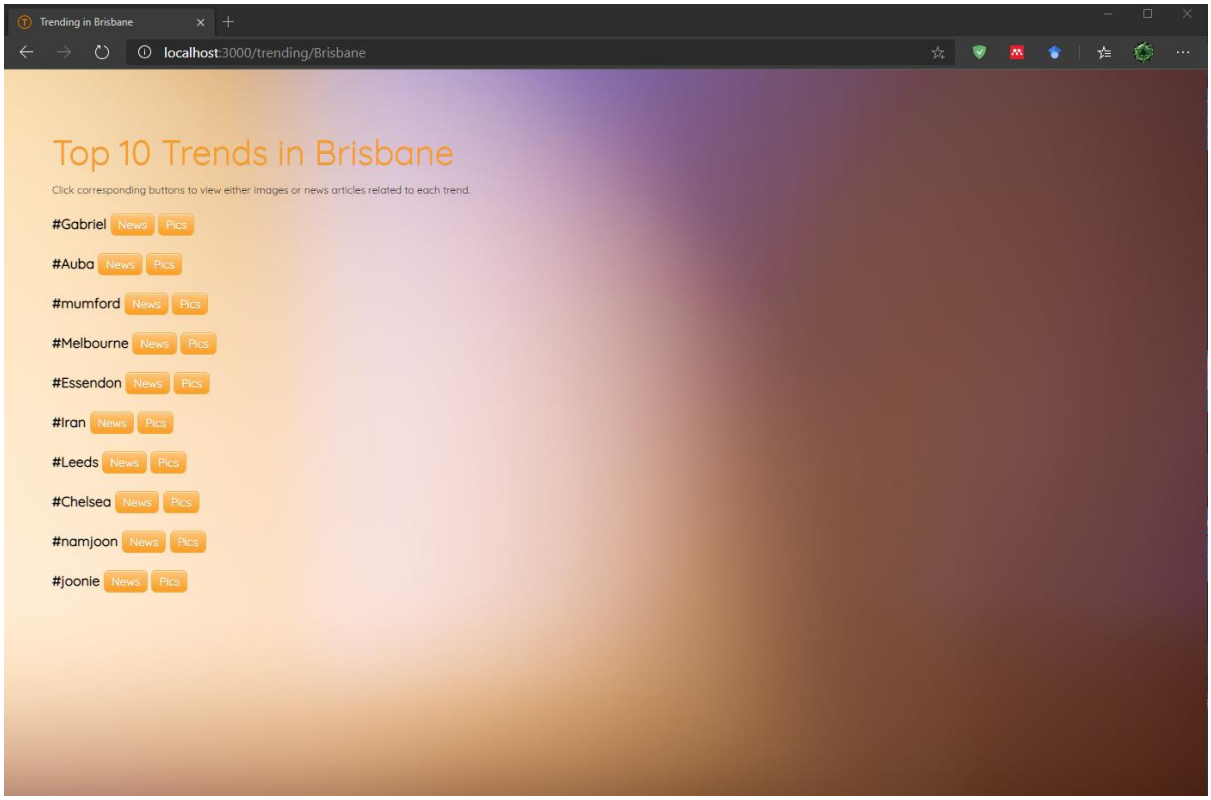
Dockerfile > ...

```
# Set node version
FROM node:dubnium

# Copy app source
COPY . /src

# Set working directory to /src
WORKDIR /src

# Install app dependencies
RUN npm install

# Expose port to outside world
EXPOSE 3000

# Start command as per package.json
CMD ["npm", "start"]
```

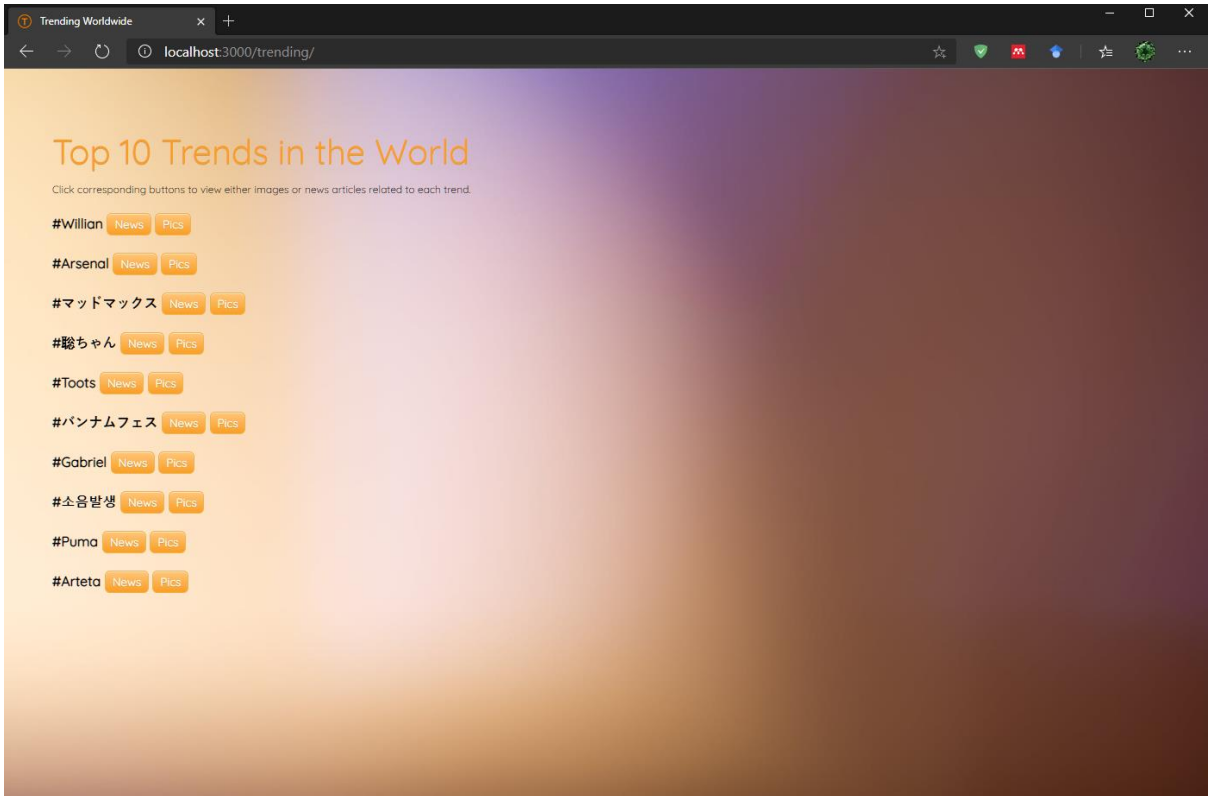
Appendix E – Screenshot result/s of test #1



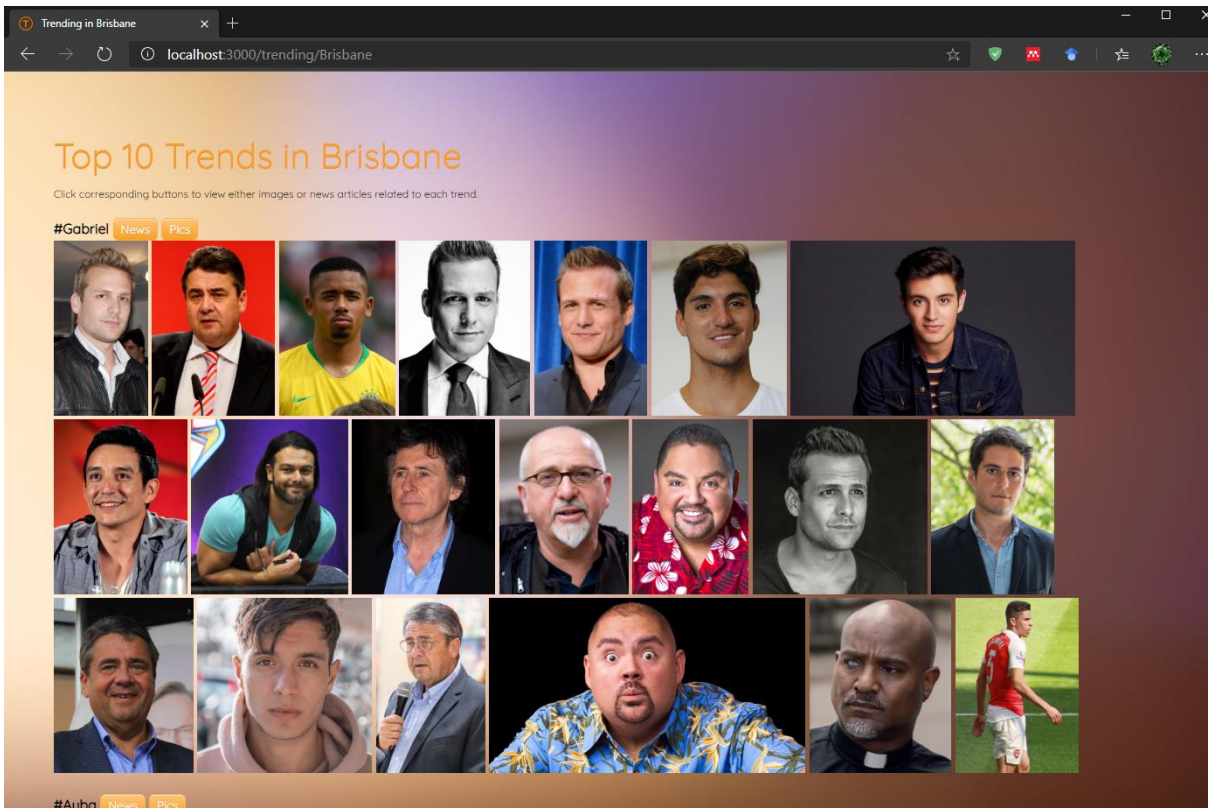
Appendix F – Screenshot result/s of test #2



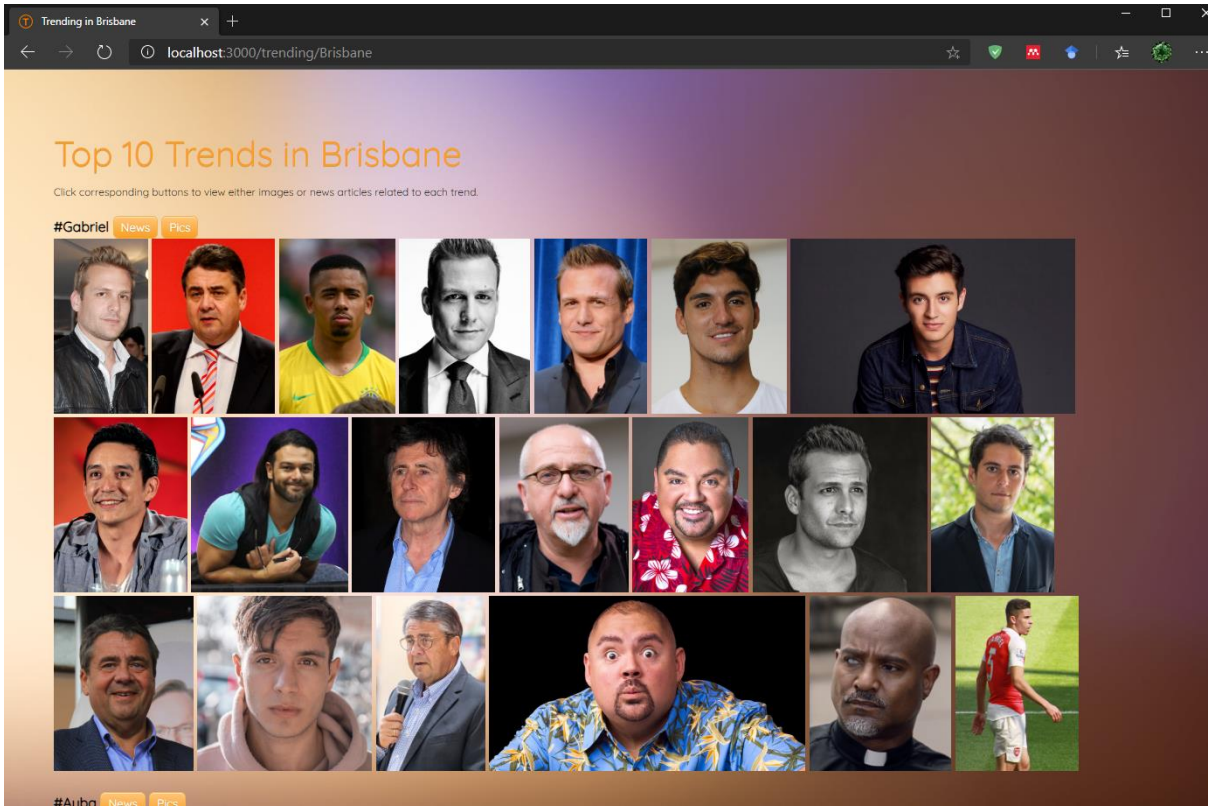
Appendix G – Screenshot result/s of test #3

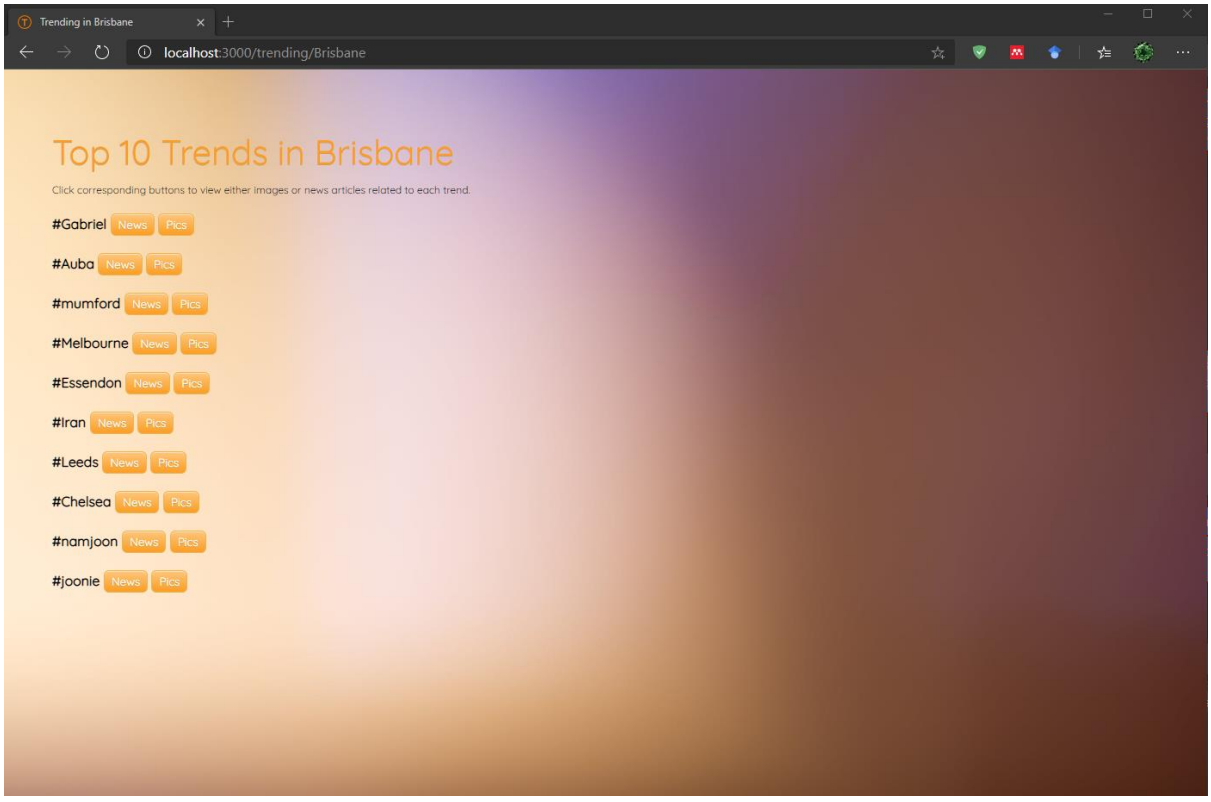


Appendix H – Screenshot result/s of test #4

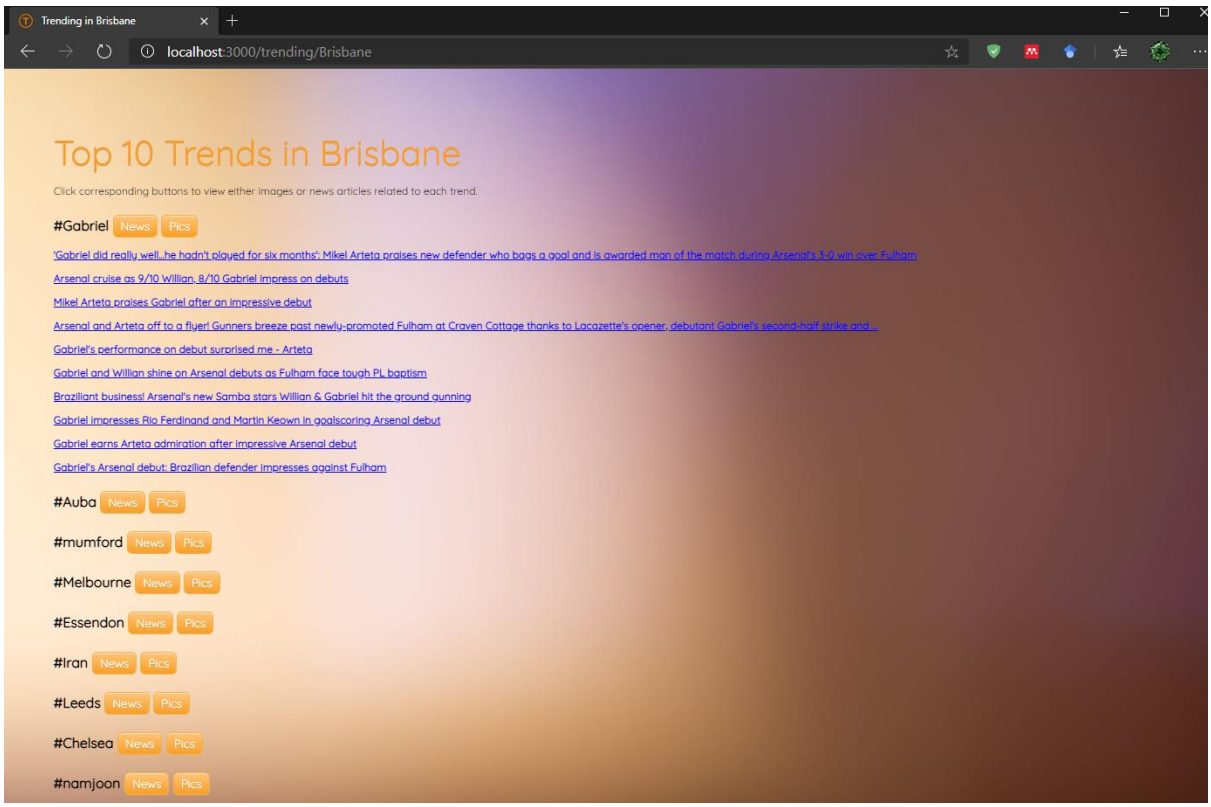


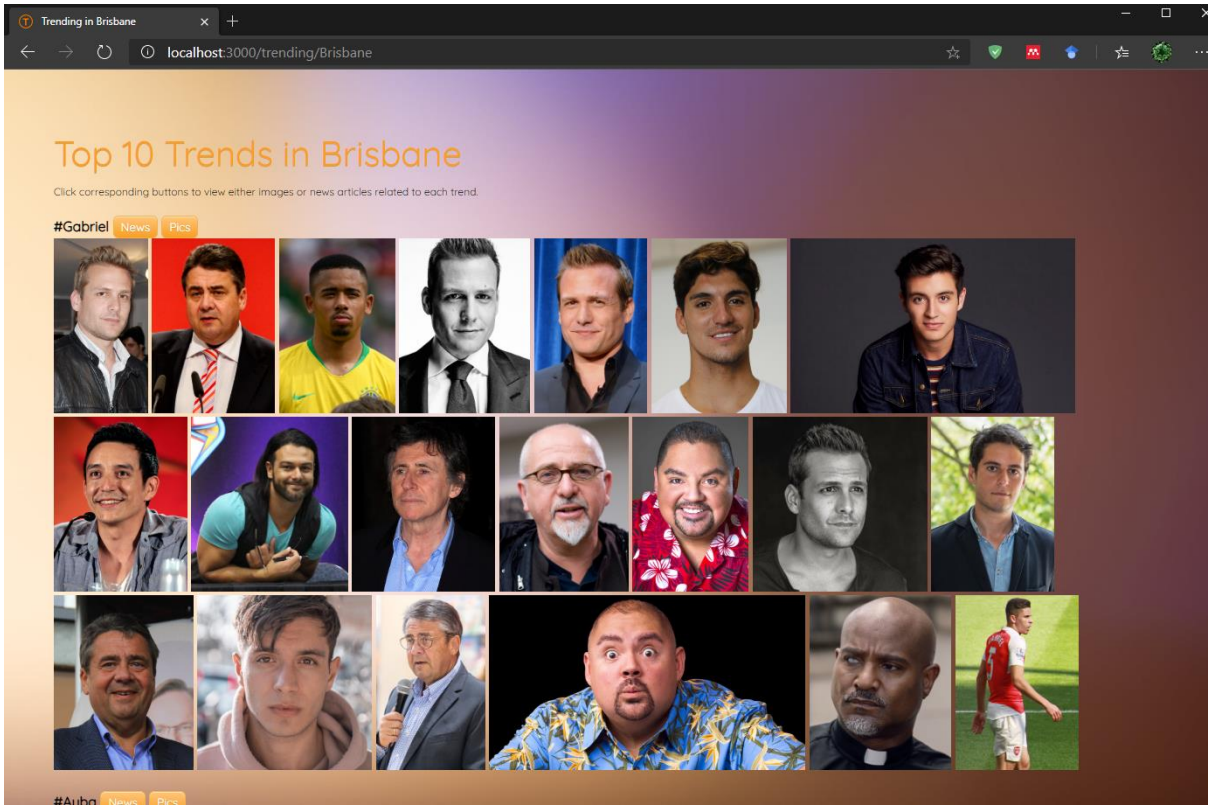
Appendix I – Screenshot result/s of test #5



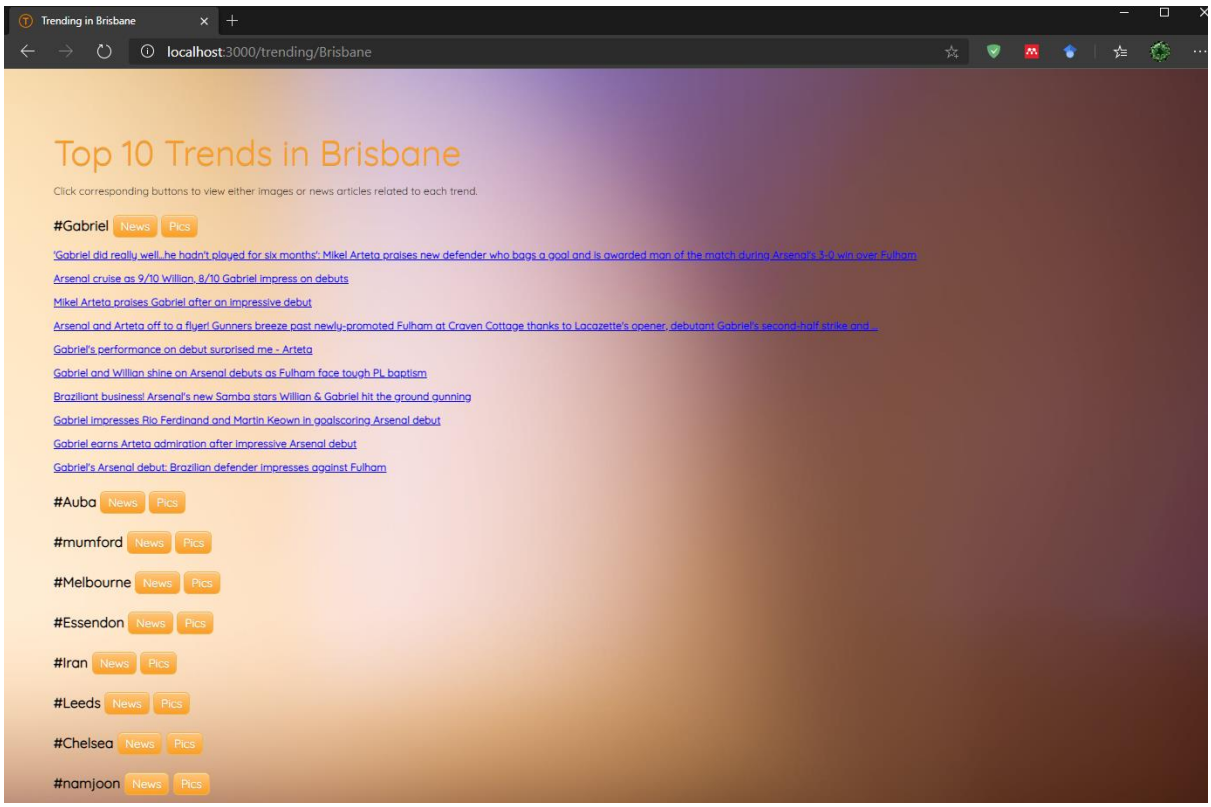


Appendix J – Screenshot result/s of test #6

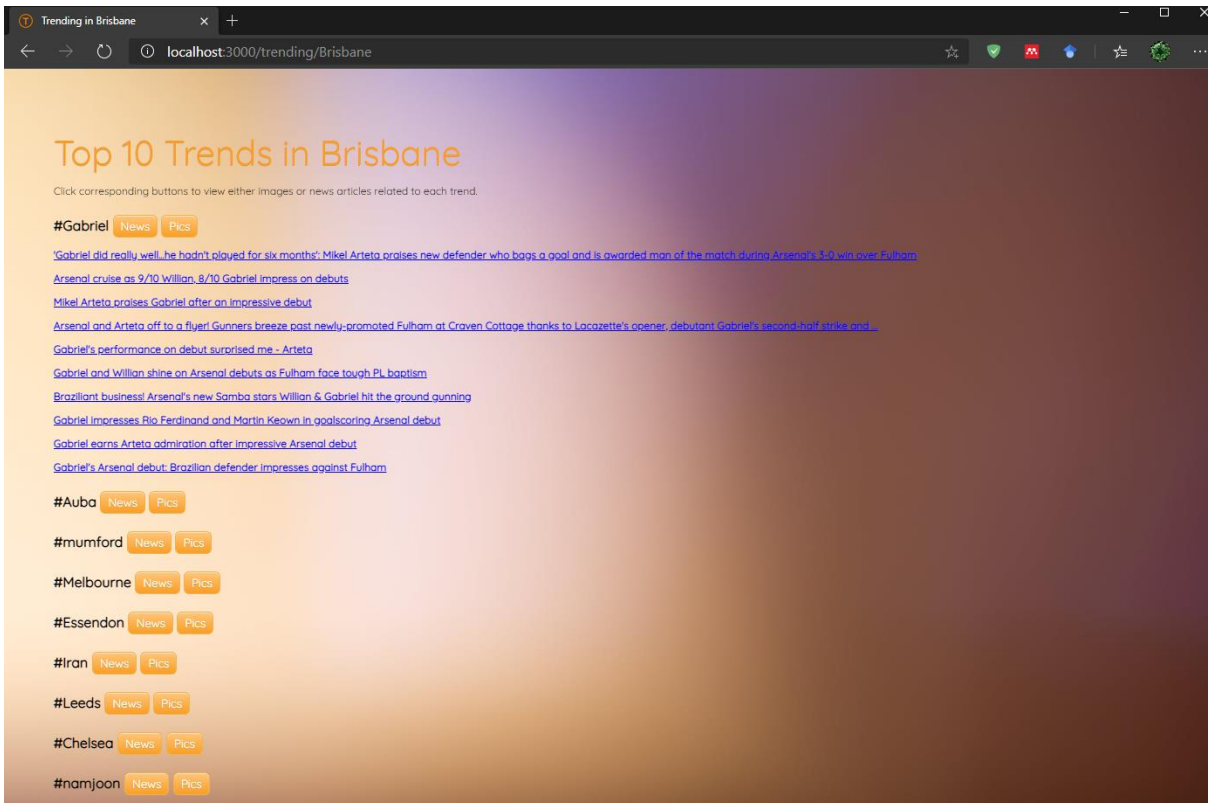


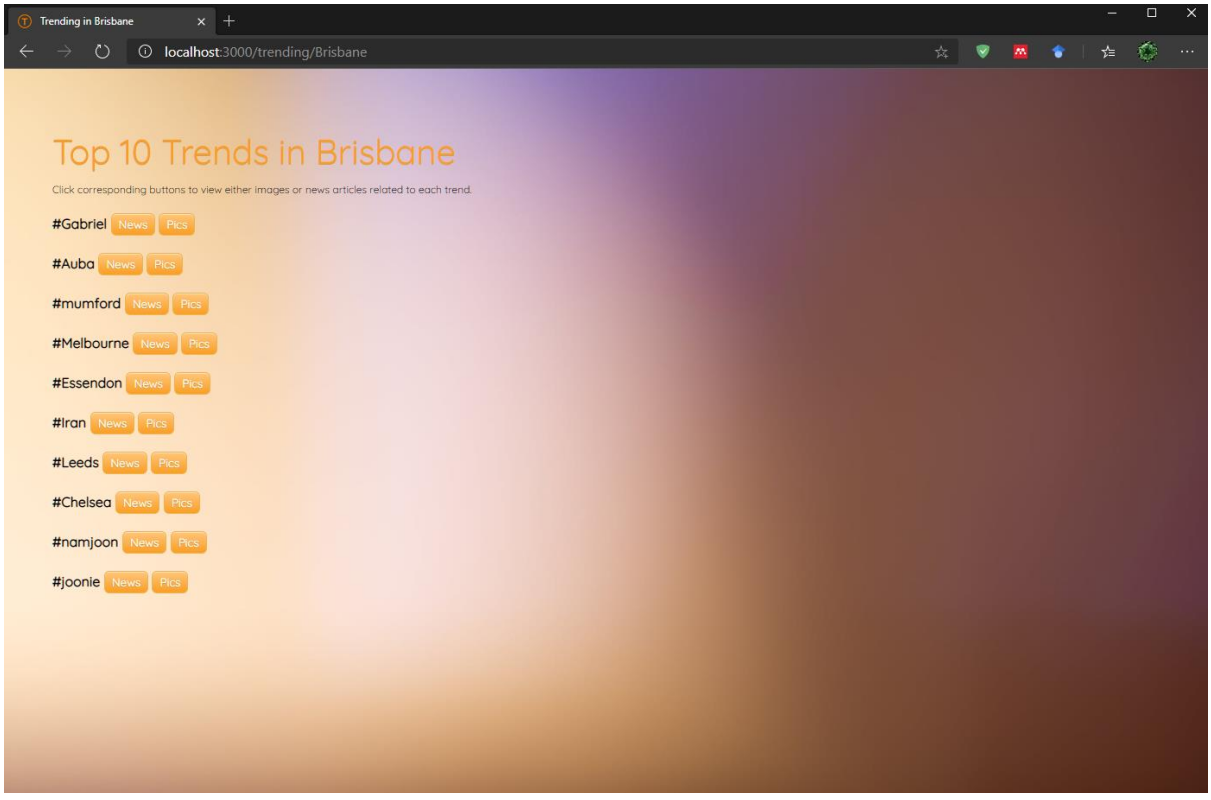


Appendix K – Screenshot result/s of test #7

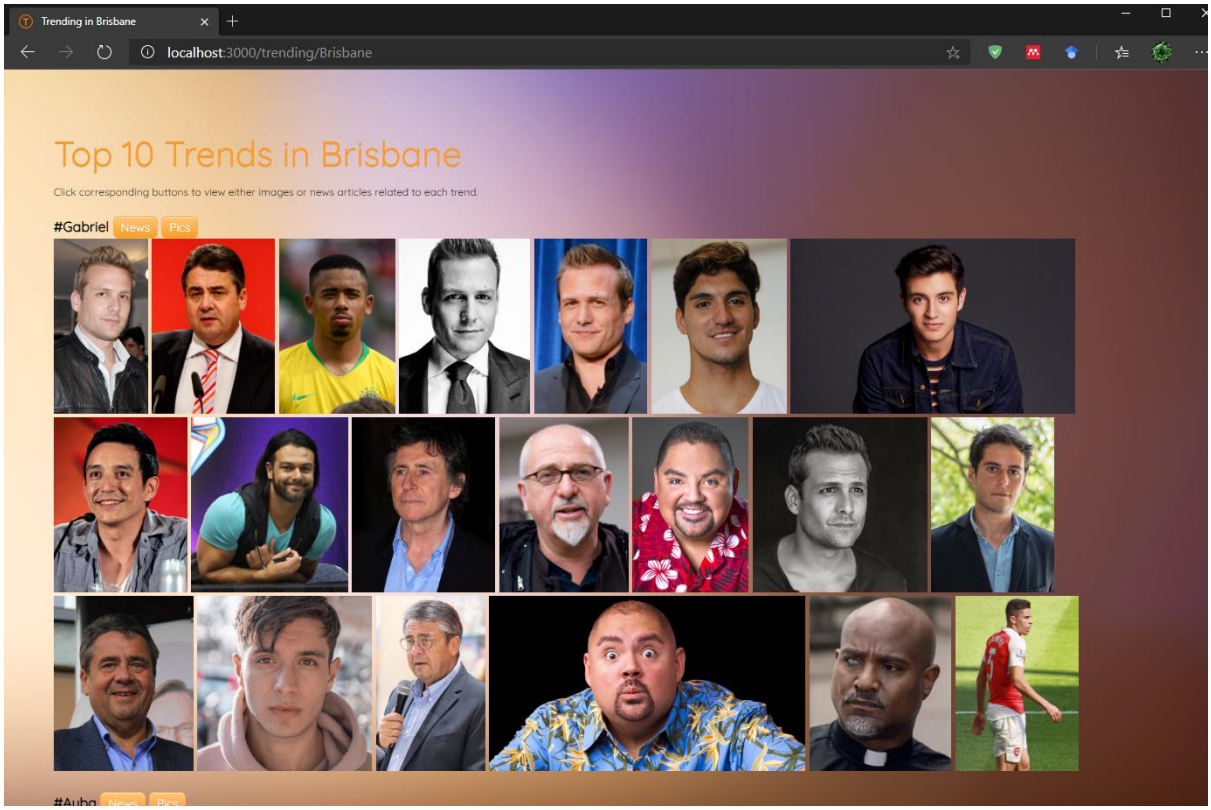


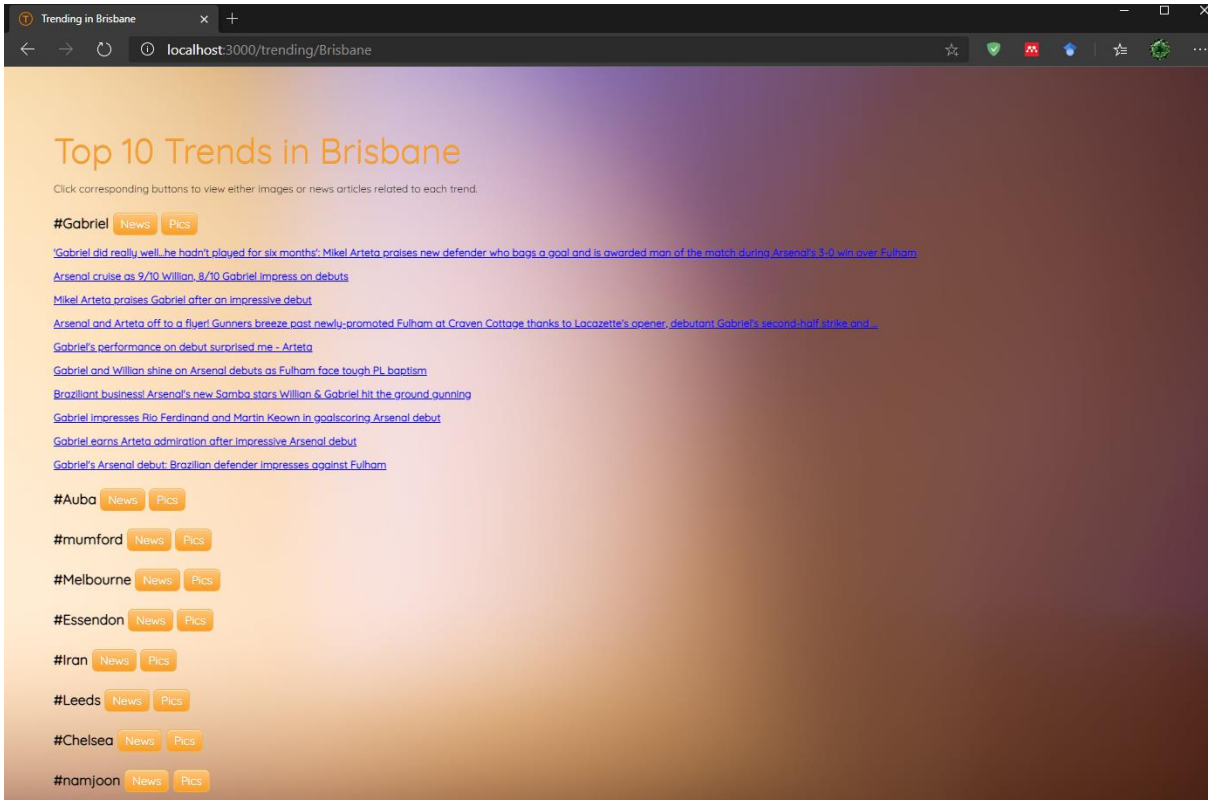
Appendix L – Screenshot result/s of test #8



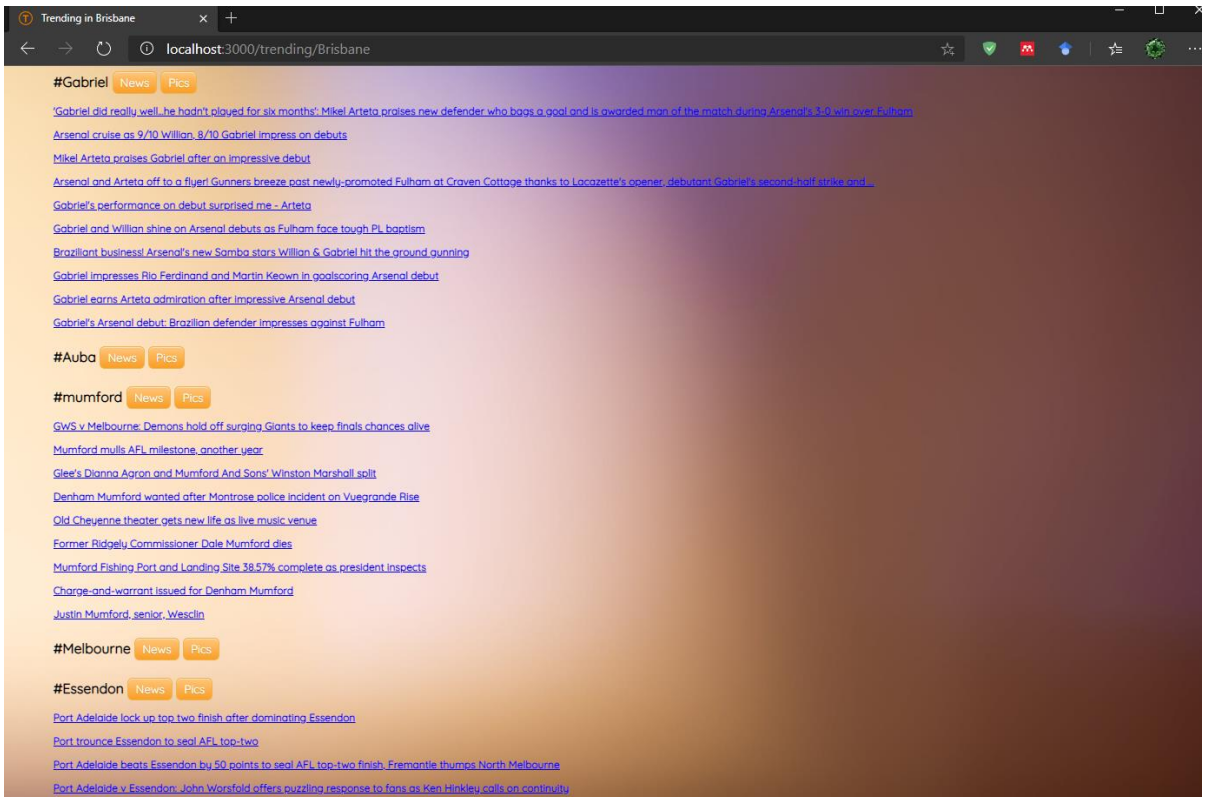


Appendix M – Screenshot result/s of test #9

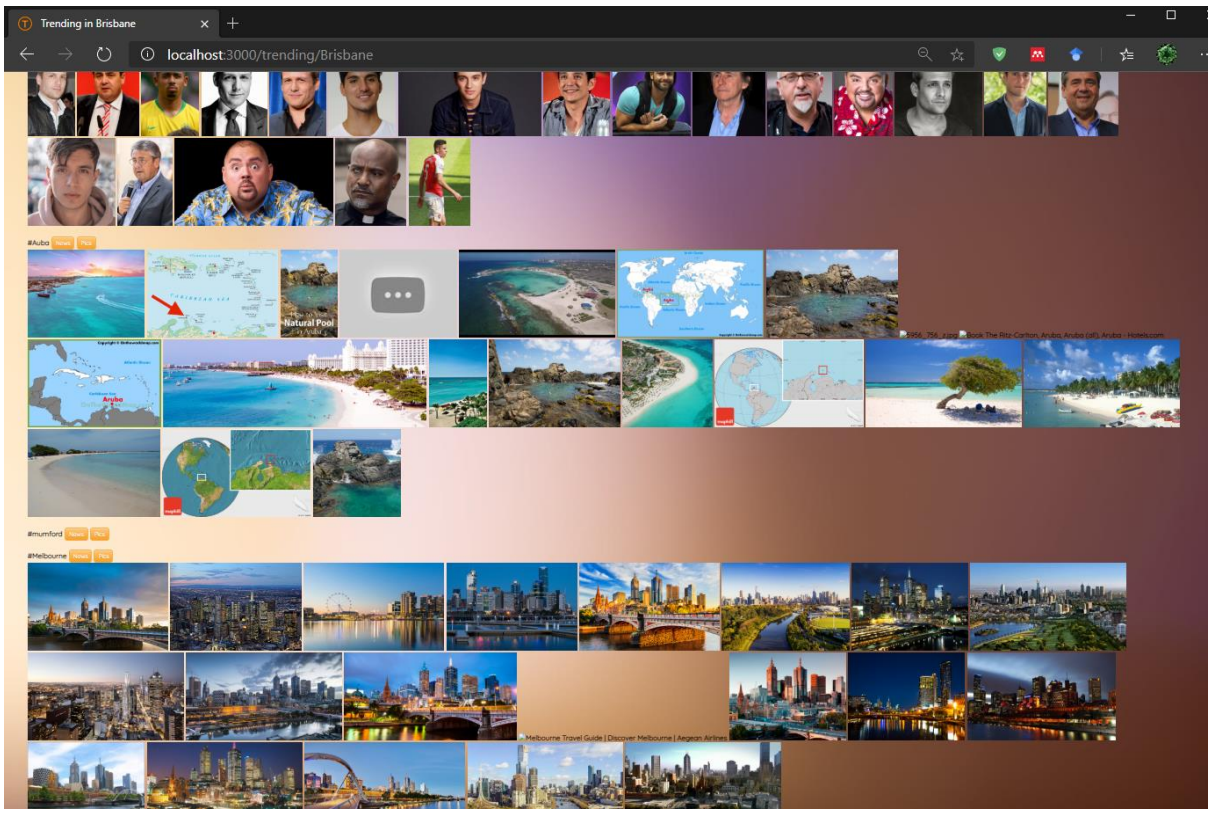




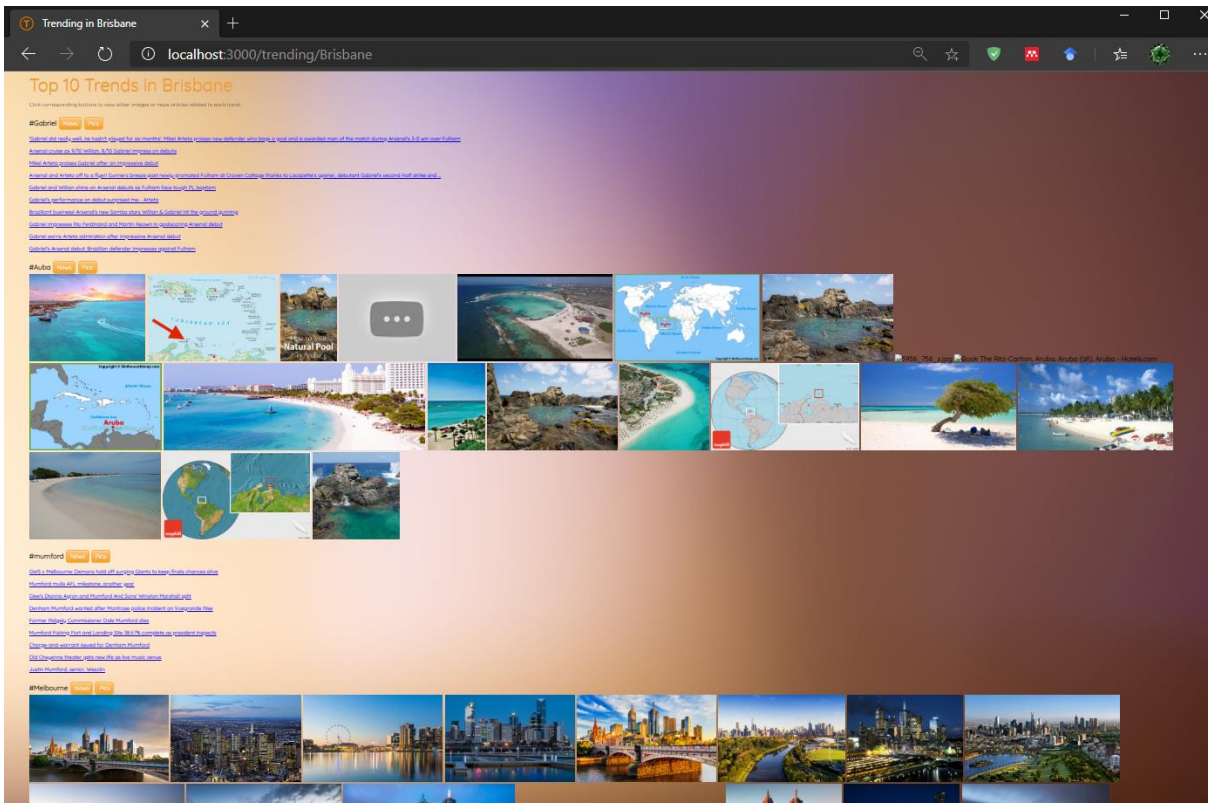
Appendix N – Screenshot result/s of test #10



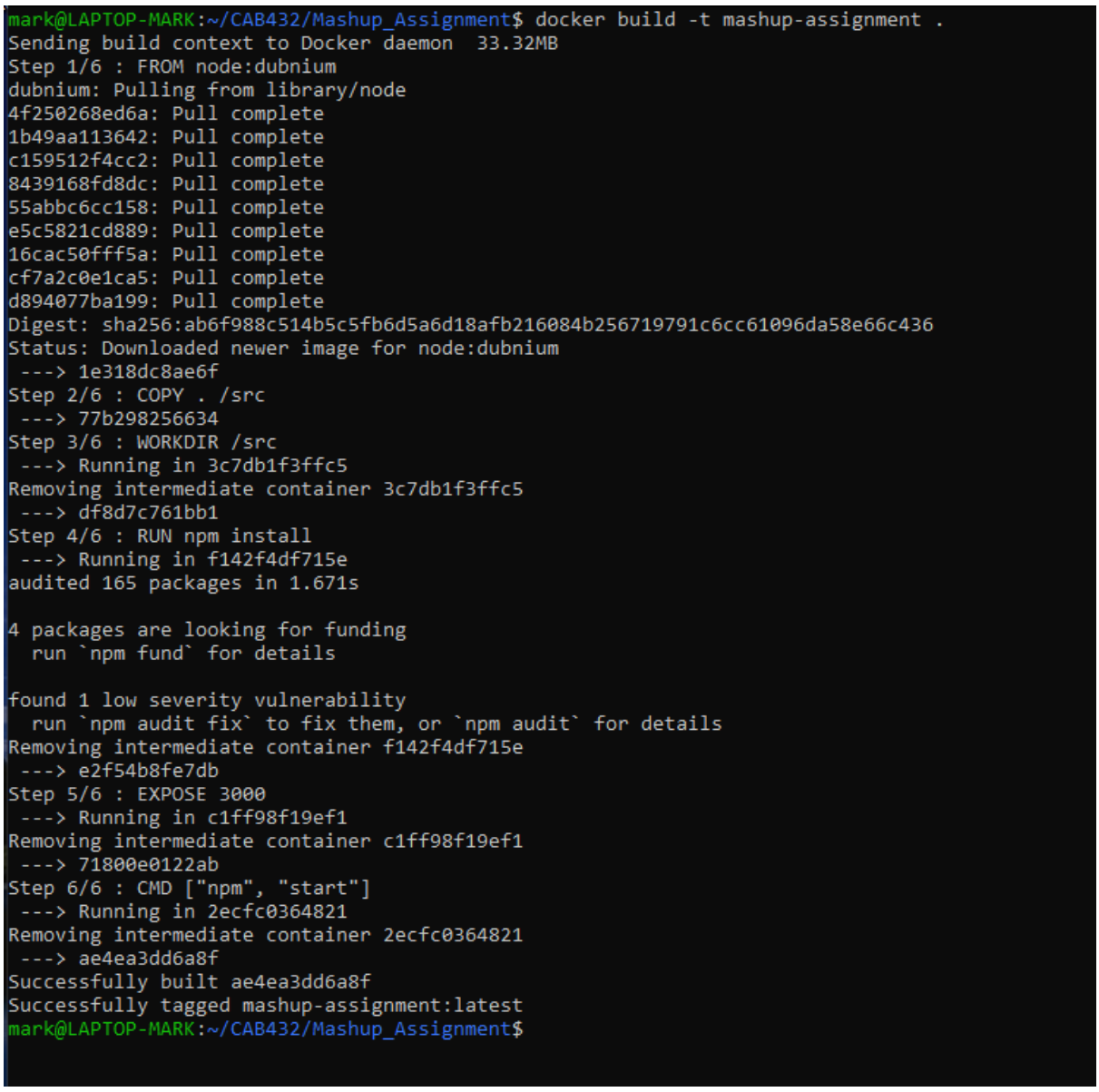
Appendix O – Screenshot result/s of test #11



Appendix P – Screenshot result/s of test #12



Appendix Q – Screenshot result/s of test #13



```
mark@LAPTOP-MARK:~/CAB432/Mashup_Assignment$ docker build -t mashup-assignment .
Sending build context to Docker daemon  33.32MB
Step 1/6 : FROM node:dubnium
dubnium: Pulling from library/node
4f250268ed6a: Pull complete
1b49aa113642: Pull complete
c159512f4cc2: Pull complete
8439168fd8dc: Pull complete
55abbc6cc158: Pull complete
e5c5821cd889: Pull complete
16cac50fff5a: Pull complete
cf7a2c0e1ca5: Pull complete
d894077ba199: Pull complete
Digest: sha256:ab6f988c514b5c5fb6d5a6d18afb216084b256719791c6cc61096da58e66c436
Status: Downloaded newer image for node:dubnium
 ---> 1e318dc8ae6f
Step 2/6 : COPY . /src
 ---> 77b298256634
Step 3/6 : WORKDIR /src
 ---> Running in 3c7db1f3ffc5
Removing intermediate container 3c7db1f3ffc5
 ---> df8d7c761bb1
Step 4/6 : RUN npm install
 ---> Running in f142f4df715e
audited 165 packages in 1.671s

4 packages are looking for funding
  run `npm fund` for details

found 1 low severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container f142f4df715e
 ---> e2f54b8fe7db
Step 5/6 : EXPOSE 3000
 ---> Running in c1ff98f19ef1
Removing intermediate container c1ff98f19ef1
 ---> 71800e0122ab
Step 6/6 : CMD ["npm", "start"]
 ---> Running in 2ecfc0364821
Removing intermediate container 2ecfc0364821
 ---> ae4ea3dd6a8f
Successfully built ae4ea3dd6a8f
Successfully tagged mashup-assignment:latest
mark@LAPTOP-MARK:~/CAB432/Mashup_Assignment$
```

Appendix R – Screenshot result/s of test #14

```
mark@LAPTOP-MARK:~/CAB432/Mashup_Assignment$ docker run -d -p 80:3000 mashup-assignment
b381b3f93e627b7619aeff0989bf37134f67b859927365c79e5b3f7086cd53ba
mark@LAPTOP-MARK:~/CAB432/Mashup_Assignment$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                  NAMES
b381b3f93e62        mashup-assignment   "docker-entrypoint.s…"   7 seconds ago       Up 5 seconds
0.0.0.0:80->3000/tcp   ecstatic_dewdney
mark@LAPTOP-MARK:~/CAB432/Mashup_Assignment$
```