Big Data Visualization using Commodity Hardware and Open Source Software – Data Visualization Component

Dave Cloete

1573016

4 July 2016

Abstract

This report focus on the design and development of the Twit-Con-Pro Data Visualisation Component.

The component is a stand-alone web server component using the Node.js Express Framework combined with web-based technologies such as HTML, JavaScript, D3 (Data Driven Documents) and Bootstrap. The solution only depends on Node.js to be installed, can run on any Node.js supported platform (Windows, Linux, or Mac), as an Internet facing or stand-alone web application.

The component visualises sentiment around a specific topic from data originating from Twitter. All software used by the component is free and the component can run on commodity hardware such as the Raspberry-PI.

Contents

1. Introduction

Data Visualisation is key to understanding large sets of data. Graphically representing data allows the human mind to better comprehend abstracted views and makes it easier to identify trends, patterns and anomalies not easily identifiable when looking at data in more traditional forms such as spreadsheets.

The Twit-Con-Pro Data Visualisation Component does just that. It essentially translate data from the Data Processing Component into vector graphics through various graphs. The types of charts chosen are:

- Word Cloud
- Bar Charts
- Streamgraphs
- Heat Maps

Furthermore, this is all done using open-source software such as Node.js, D3, and commodity hardware, a Raspberry-PI 3.

2. Background

One of the Twit-Con-Pro project's primary focuses where to prove that big data processing and visualisation is possible using free open-source software combined with inexpensive hardware and not reserved for the enterprise domain.

Twit-Con-Pro sources data from Twitter and attempts to visualise sentiment around a topic, the specific topics chosen for this project was US and SA Elections. Categories representing the political parties and candidates were defined in order to visualise the positivity and/or negativity around each category as well as the combined sentiment across the categories.

Initially the decision was made to do small POCs (Proof of Concepts) to understand which technologies can be used to develop a data visualisation solution on the Raspberry-PI hardware. Investigation shown various options existed on the latest Raspberry-PI 3 using the Raspbain OS which is a flavour of Debian part of the Linux Operating systems family.

3. Requirements

The following is a list of the key requirements.

3.1. Functional

- Represent the total amount of tweets mentioning a category in comparison
- Represent data over time in daily and hourly increments
- Represent the sentiment for a category in a positive and negative light (Con-Pro)

3.2. Non-functional

- The concept of representing sentiment should be generic, in other words, the solution should be able to represent topics other than the US and SA Elections
- The charts should be scalable / responsive
- The charts should be represented side by side so that comparisons can be drawn across charts
- The solution had to run on commodity hardware such as the Raspberry-PI
- Data was to be present in such a way that comparisons can be drawn

4. Approach

Various POCs were developed to both learn technologies and determine viability.

The approach taken to develop the Data Visualisation Component was iterative in nature. While the framework for the component was built in a single release, each Graph View started as a simple prototype and gained form over various iterations after weekly review sessions.

From a User Experience Point of view, the approach was simplicity, 'keep it simple'. Focus was placed on a responsive and easily navigable user interface. Each graph view was tested on 3 different viewpoints ranging from a resolution of 550x760pixels up to 1920x1080pixels including catering for devices supporting screen rotate.

5. Challenges

Project challenges included collaboration and communication. These were mitigated early on with weekly project meetings and online collaboration tools.

By far the greatest data visualisation specific challenge faced was learning the D3 component. D3 was chosen for its versatility and ability to handle complex datasets and inheritably required a deep understanding of digesting complex data sets into various SVG element and attributes. D3 contains power data processing functions such as map reducing, nesting and scalar type functions returning other functions or data allowing the translation of domain data into graphics.

6. Design Overview

This section describes how the Data Visualisation Component was designed starting with a conceptual view, technologies used followed by describing each of the sub components.

6.1. Conceptual

The Data Visualisation Component comprises of two parts:

- The Framework – Responsible for hosting the Graph Views based on a set of configuration Models.
- The Graph Views – Responsibly for representing the data provided by the Data Processing Component given the configuration defined by the Framework.

Both the Framework and Graph views are strutted similarly in an MVC implementation.

The MVC pattern was considered and implemented as it provided a method of modularisation and separation of concerns. The Data Visualisation Component was thus split into the three main modules, Views, Controllers and Data Models.
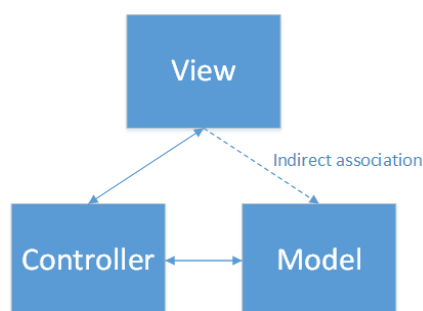

Figure 1 – Depiction of the MVC pattern

Applying the MVC pattern also allowed for one controller method to be utilised by many views and one view to utilise many controller methods thus maximising reusability and portability of

components. The separated concerns means that the views component was not dependant on how the controller logic acquired data, only that the controller logic can acquire data of a consumable format. This further supported Software engineering principals like Coupling and Cohesion.

The Visualisation Framework itself followed the same MVC pattern. Configuration files were created to further decouple views/graphs from the data sources. These configuration files can be seen as the Framework's Data Models.

6.2.   Technologies Used
Node.js

Node.js is one of the technologies shipped with Raspbian however an update was required as the version was quite old. Node.js (1), built on the Google Chrome's V8 (2) engine allows for powerful JavaScript stand-alone applications to be host locally or on a server.

Node.js was also chosen as this meant the entire Data Visualisation Component (browser and server-side components) could be written without using too many programming languages, beneficial from an academic point of view as it meant less languages to learn.

Node.js – Express Framework

The Node.js Express Framework (3) was chosen as the server-side technology as it allows for serving both static and dynamic content with little coding necessary.

There are many ways to write a web application in Node.js, the Express Framework is merely one of the frameworks making this possible and within the Express Framework, there's more than one way of writing a web application.

The Express framework supports a MVC implementation. One of the supporting components in the Express Framework is the Router component allowing the HTTP requests to be handled specific functions based on the specified request path/route. Routes were logically grouped into script files.

Node.js – File System

Twit-Con-Pro uses JSON files as a standard for transferring data between the components. JSON files were thus chosen as the storage medium for the Data Models as it meant no additional complexity were required for storing data.

Node.js ships with a module called 'fs' allowing for the reading and writing of the data model JSON files.

Node.js Twig.js

Twig.js was chosen as a HTML templating tool to allow the controllers to embed data when rending the views.

Twig (4) is a PHP based HTML templating engine and was ported to Node.js as an Open Source Initiative.

Twig.js was considered over other components such as Vash and Bliss as it is more popular based on download history and the team had prior Twig experience.

Bootstrap

Bootstrap is an Open Source Initiative and was chosen to allow for a responsive UI. This allowed for creating a response base without having to specifically code for it (5).

Bootstrap Panels were used as a container for the Graph Views. These panels are then placed on the dashboard using a Bootstrap Grid allowing for panels to respond to different viewpoints to further aid responsiveness.

D3

D3 (Data Driven Documents) was chosen for its versatility and ability to handle complex datasets. D3 works with standard web technologies such as HTML, SVG, CSS and JavaScript allowing for powerful, interactive data visualisations (6).

D3 made possible the translation of domain data into scalable vector graphics.

All charts except for the Word Cload were generated using the D3 component. Common functions includes 'scale.liniar' and 'scale.ordinal' aiding generation of X- and Y-Axis labels, and the 'select' and 'data' function combinations which created elements based on the dataset provided.

Other technologies considered were Chart.js, FusionCharts and InfoVis however did not provide the flexibility that D3 offered.

Wordcloud2.js

Wordcloud2.js (7) was chosen due to its simplicity. Wordcloud2.js produces a 2 dimensional set of words from a list of words and a count value where the word with the highest count, has the largest font size.

Word clouds are typically used to visually represent popular words in a context. In context of Twit-Con-Pro this was chosen to better understand if there were a set of specific words that could better provide context around the sentiment.

6.3. Physical Project Structure

The Data Visualisation Component is a stand-alone web application. To best describe the physical structure one can look at the folder structure of the application.
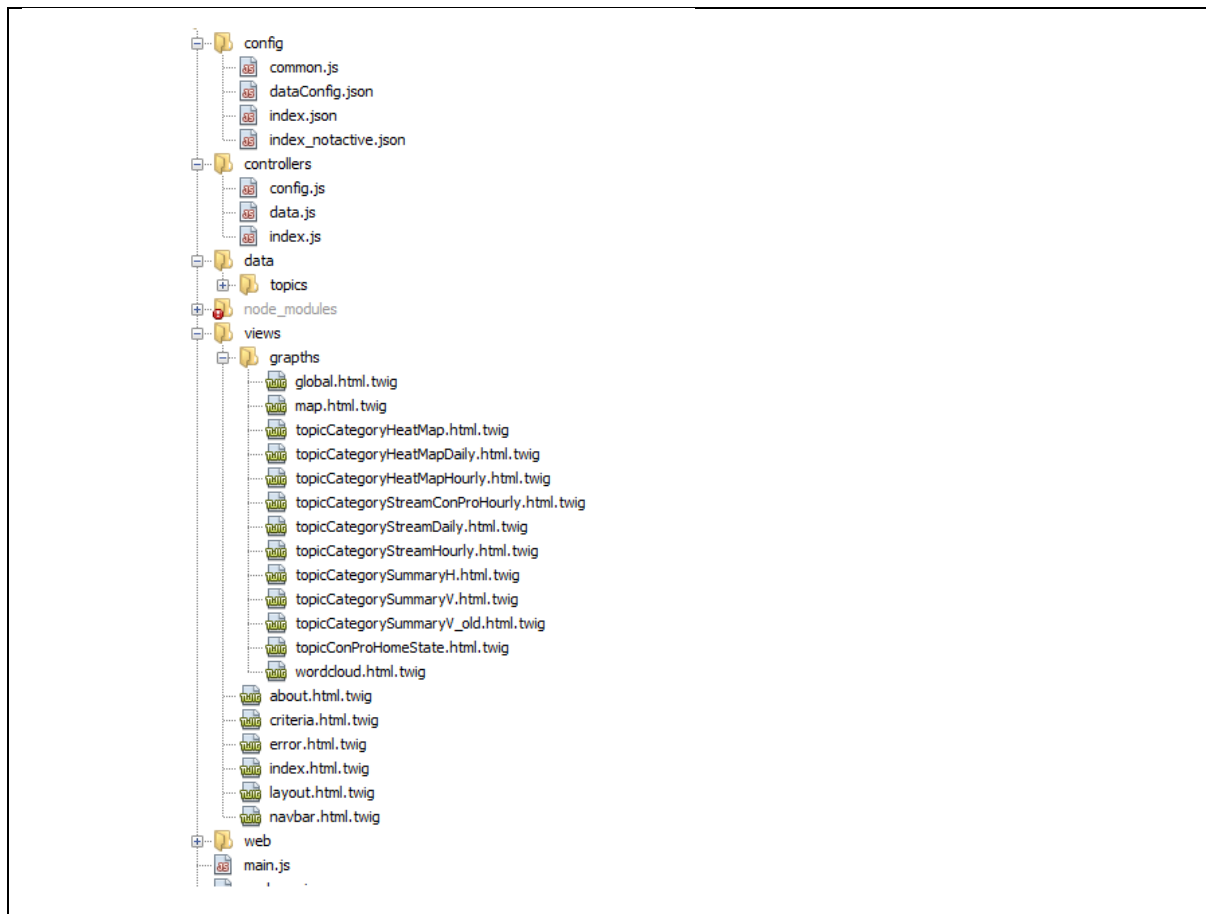

Figure 2 – Snippet from NetBeans Project Explorer View

The main.js file contained in the root of the application is the script executed at start up. This script instantiates the project, loading the Express Framework and then listens on the specified port for HTTP request.

The MVC implementation can be seen by observing the folder structure containing a separate sub folder for 'data'(Data **Model**), 'views' (**View**) and controllers (**Controller**).

One exception is the Framework's Data Model is not contained in the 'data' folder and rather placed in the 'config' folder as the Graph View's Data Model root location is a configurable value to allow for different applications of the Data Visualisation Component.

6.4. The Framework

The Framework consists primarily of a Dashboard component and a settings component.

Dashboard

The dashboard component is loaded when a web client browser request to load the site by navigating to http://localhost:8081 (given the component is hosted on the local machine). The Index Controller handles this request by loading the index.json Data Model which contains the Panel configuration.

An entry in the Panel configuration represented a graph view and its properties including which controller method it should use to retrieve its data from.

```
"panels": [
    {
        "name": "Word Cloud",
        "size": "3",
        "type": "grapths/wordcloud.html.twig",
        "dataUrl": "data/wordcloud",
        "specialClass": "panel-info"
    },
```

Figure 3 - Snippet from the index.json Data Model

The Index Controller will then fetch the topic configuration to retrieve the topic name for display purposes. These Data Models are then bound to the View by calling the Twig.js rendering component within the HTTP response builder method 'res.render'.

The Index View template contains twig functions to render a panel containing each of the Graph Views configured in the index.json Data Model.

```twig
<div class="row">
    {% set sizeTotal = 0 %}
    {% for panel in data.panels %}
        {% if sizeTotal >= 12 %}
        </div>
        <div class="row">
            {% set sizeTotal = 0 %}
        {% endif %}
        <div class="col-md-{{panel.size}} col-sm-{{panel.size * 2}}">
            <div class="panel {{ panel.specialClass is defined ? panel.specialClass : "panel-default"}}">
                <div class="panel-heading">
                    <h3 class="panel-title">{{panel.name}}</h3>
                </div>
                <div class="panel-body">

                    {% include panel.type with { 'dataUrl': panel.dataUrl } %}

                </div>
            </div>
        </div>

        {% set sizeTotal = sizeTotal + panel.size %}
    {% endfor %}
</div>
```

Figure 4 – Partial Code Snippet from the index.html.twig View

Twig provides the ability to inherit and/or include other HTML templates. This allowed for other components to be included into the dashboard in such a way they these components can be reused in other views. An example of such reuse can be found in the 'Info' view when selecting the Info option in the top right of the dashboard.

The Navigation Bar itself is a partial view and not dependant on the dashboard view.

Settings

The Data Visualisation Component was designed to handle multiple topics. The user can select a topic from a configured set of topics by selecting the Settings option ⚙ in the top right of the dashboard.
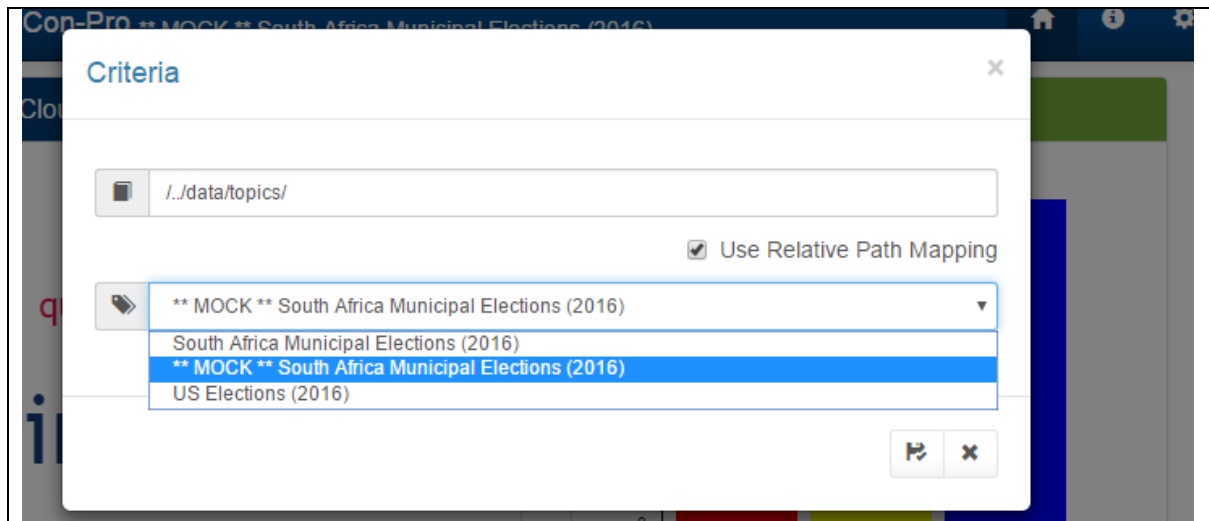

Figure 5 – Sample of the Settings Popup View

As seen above, there 'data' folder location is also configurable as the Data Visualisation Component was designed to be agnostic of the content, it can show sentiment of topics other than the specified Election based topics.

6.5.   Data Models

All Data Models are contained in JSON files. The Data Models are essentially the output of the Data Processing Component

Category Color


```
    "Category": "EFF",
    "Color": "#FF0000",
    "ConColor": "#440000"
```

Figure 6 – Snippet from the CategoryColors.json Data Model

Field	Description
Category	The Name of the category
Color	The primary colour to represent the category
ConColor	The colour to use represent negative sentiment for the category

Figure 7 – Field Description

Category Count per Day and Category Count per Hour

```
{
    "Date": "2016-05-01T00:00:00Z",
    "Data": [
        {
            "Category": "EFF",
            "Count": 10
        },
        {
            "Category": "ANC",
            "Count": 15
        },
        {
            "Category": "DA",
            "Count": 5
        }
    ]
}
```

Figure 8 – Snippet from the CategoryCountPerDay.json Data Model

Field	Description
Date	Date and time of the collected data
Data	Array containing an entry for each category
- Category	The Name of the category
- Count	Amount of Tweets mentioning the category within the time period

Figure 9 – Field Description

Category Summary

```
{
    "Category": "ANC",
    "Pro": 870,
    "Con": 200,
    "Count": 1070
}
```

Figure 10 – Snippet from the CategorySummary.json Data Model

Field	Description
Category	The Name of the category
Pro	Total Amount of Tweets mentioning the category positively
Con	Total Amount of Tweets mentioning the category negatively
Count	Total Amount of Tweets mentioning the category

Figure 11 – Field Description

Con-Pro Count per Hour

```
{
    "Date": "2016-05-01T05:00:00Z",
    "Data": [
        {
            "Category": "EFF",
            "ProCount": 5,
            "ConCount": 15
        },
        {
            "Category": "ANC",
```

Figure 12 – Snippet from the ConProCountPerHour.json Data Model

Field	Description
Date	Date and time of the collected data
Data	Array containing an entry for each category
- Category	The Name of the category
- ProCount	Amount of Tweets mentioning the category positively within the time period
- ConCount	Amount of Tweets mentioning the category negatively within the time period

Figure 13 – Field Description

Word Count

```
},
{
    "word": "money",
    "count": 44
},
{
    "word": "fox",
    "count": 20
},
```

Figure 14 - Snippet from the WordCount.json Data Model

Field	Description
Word	The word
Count	Amount of times the word appear in the entire dataset

Figure 15 – Field Description

6.6.   Controllers

As the function of the Framework is covered in section 5.4, this section will focus on the Data Controller.

The Data Controller is responsible to provide the Graph Views with the requested Data Models. Each of the controller methods follow exactly the same implementation with only the route and JSON file name that differs.

```
router.get('/categoryCountPerDay', function (req, res) {
    config.getDefaultTopicDataFile('CategoryCountPerDay.json', function (data) {
        res.setHeader('Content-Type', 'application/json');
        res.send(data);
    }, function () {
        res.send(null);
    });
});
```

Figure 16 - Snippet from the data.js Controller

Figure 17 is an example of a Data Controller Method.
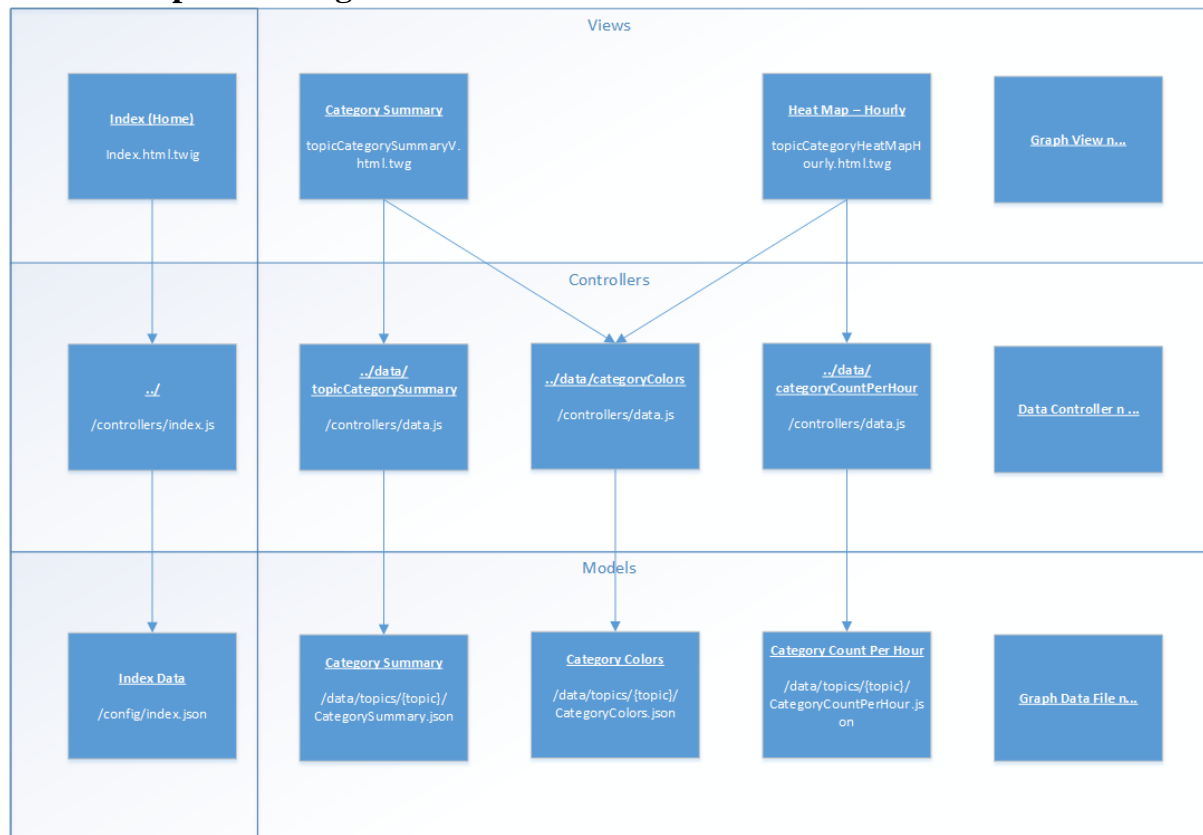
6.7.   Component Diagram

Figure 17 – Data Visualisation Component Diagram

6.8.   Graph Views

Each Graph View consists of a CSS, HTML and JavaScript component. For simplicity, all three components were placed in a single twig template. Shared JavaScript resources were combined and placed in a static JavaScript file located in the 'web/js/charts/common.js' file.

There are 10 different Graph Views, all structured in a similar way:

As explained in section 5.4, the Graph Views are rendered within the Index View where the configured Controller method is injected into the Graph view.

```
<script type="text/javascript">
    $(document).ready(function () {
      var dataUrl = "{{dataUrl}}";
```

Figure 18 - Snippet from topicCategorySummaryV.html.twig View template showing how the Controller Method is injected into the view

The D3 Queue (8) component was used to retrieve Data Models asynchronously

```
queue()
  .defer(d3.json, dataUrl)
  .defer(d3.json, "/data/categoryColors")
  .await(function (error, data, categoryColors) {
    if (error) {
      $('#hm_dataPlaceholder').html("error: " + error);
      return;
    }
    $('#v_dataPlaceholder').html("");
    $('#v_dataPlaceholder').height(250);
    basicVBarChart.initialise(data, categoryColors, '#v_dataPlaceholder');
  });
});
```

Figure 19 - Snippet from topicCategorySummaryV.html.twig View showing how the queue D3 queue component is used to asynchronously load two Data Models.

After all the data has been loaded the chart will be initialised by calling the chart object's initialise method.

Each Chart Object function sets differs depending on the steps required to draw the specific chart however each chart has at least two functions:

- '.initialise' – Used to prepare the graph area before the '.draw' function is call. Another function of the '.initialise' function is to ensure the graph area is cleared and ready to be redrawn after a window resize occurs. This allows for the graphs to be responsive.
- '.draw' – Used to draw the graph on the canvas given the data received.

6.9. Testing

A mock Data Model set, "sa-2016-Mock", was created to represent each data source to allow for testing of the Graphs. The mock dataset included data to test specific scenarios as to ensure a proper contract is given.

Responsive UI testing was performed by resizing the browser windows and by making use of Google Chrome Developer Tools' device toggle functionality.

Testing was performed by the developer and during the weekly team meeting, the functionality was demoed to the team and the feedback was incorporated.

Towards the end of the project load tests were executed using WAPT (9), a web-site load testing tool. Please see Appendix D for detail.

7. Recommendations

The Data Collection Component collects geo-location data as originally Chart concepts such as a global heat map was identified as a very informative view especially considering Elections as a topic. The source code contains two prototypes of such charts using mock data.

The Data Controller methods could have been made more generic as each method does exactly the same thing. There should have rather been one method accepting the Data Model type as a parameter and the controller method should have logic to determine which Data Model to load. This would make the Data Controller more easily extendable.

As the Graph Views are compiled into the Index View they share the DOM with each other. For this purpose each Graph View contained a HTML <DIV> or <SVG> element with a unique id. This approached worked to a large extent however is not scalable. A better implementation would have been a widget based concept, using JQuery UI Widget Factory.

Graph View styles were mostly class based and thus allowing for one Graph's style section to affect another graph. Two possible options exist, combining the styles into one global stylesheet or adding unique classes for each style.

8. Conclusion

The project successfully succeeded in providing a low cost solution for Big Data Processing and Visualisation using open source software on commodity hardware.

The solution can potentially be beneficial to learners in the academic space as well as small businesses that can benefit from a big data solution without having to invest a large amount is infrastructure and/or licencing.

9. References

1. **Node.js Foundation.** Home. *Node.js.* [Online] https://nodejs.org/en/.

2. **Google.** Chrome V8. *Google Developers.* [Online] https://developers.google.com/v8/.

3. **Wilson, TJ Holowaychuk & Douglas Christopher.** Express. *NPM.* [Online] https://www.npmjs.com/package/express.

4. **Sensiolabs.** Twig. *Twig.* [Online] http://twig.sensiolabs.org/.

5. **Bootstrap Contributors.** *Bootstrap.* [Online] http://getbootstrap.com/.

6. **D3.** Overview. *D3.* [Online] https://d3js.org/.

7. **Chien, Timothy Guan-tin.** wordcloud2.js. *Github.* [Online] https://github.com/timdream/wordcloud2.js.

8. **Bostock, Mike.** Asynchronous Queue. *Mike Bostock's Block.* [Online] http://bl.ocks.org/mbostock/1696080.

9. **SoftLogica.** Home. *WAPT.* [Online] http://www.loadtestingtool.com/.

10. **Bostock, Mike.** Let's Make a Bar Chart. *Mike Bostock.* [Online] https://bost.ocks.org/mike/bar/.

# 10.	Appendices

Appendix A: Timesheets

ELEN-7046 Group Project – Individual Time sheet - Dave Cloete					
Task	Date	Start Time	End Time	Estimated Hours	Actual Hours
Project Group Meeting 1	24 April 2016	14:30:00	16:30:00	2	02:00:00
Setup PI with Node.js and NAS mount	28 April 2016	17:00:00	21:00:00	2	04:00:00
UI POC: Node.js Static web and Controllers	1 May 2016	09:00:00	12:00:00	2	03:00:00
Project Group Meeting 2	1 May 2016	14:30:00	16:30:00	2	02:00:00
UI POC: Node.js Controllers - Added twig templating	3 May 2016	17:00:00	22:00:00	4	05:00:00
UI POC: D3 Charting - Globe Chart	7 May 2016	10:00:00	15:00:00	4	05:00:00
Project Group Meeting 3	8 May 2016	14:30:00	16:30:00	2	02:00:00
UI POC: D3 charting - Bar Chart tutorial, customised V Bar chart	14 May 2016	10:00:00	18:00:00	4	08:00:00
Project Group Meeting 4	15 May 2016	14:30:00	16:30:00	2	02:00:00
UI POC: D3 charting - Heat Maps (Days/Category)	16 May 2016	16:00:00	20:00:00	4	04:00:00
UI POC: D3 charting - Heat Maps (Days/months) + refactoring of	21 May 2016	14:30:00	16:30:00	4	02:00:00
Project Group Meeting 5	22 May 2016	14:30:00	16:30:00	2	02:00:00
UI POC: Refactor and Testing	22 May 2016	22:00:00	23:00:00	2	01:00:00
UI: Heat maps - Final	28 May 2016	10:00:00	17:00:00	8	07:00:00
UI: World /state Maps - Part 1	4 June 2016	20:00:00	22:00:00	2	02:00:00
Project Group Meeting 6	5 June 2016	12:00:00	18:00:00	2	06:00:00
UI: Criteria Configuration - Part 1	8 June 2016	18:00:00	22:00:00	2	04:00:00
Project Group Meeting 7	11 June 2016	10:00:00	19:00:00	2	09:00:00
UI: Criteria Configuration - Part 2	12 June 2016	19:00:00	23:00:00	4	04:00:00
UI: Initial Stream Graph Research	14 June 2016	19:00:00	23:00:00	4	04:00:00
UI: Stream Graphs - Part 1	16 June 2016	10:00:00	18:00:00	4	08:00:00
UI: Stream Graphs - Part 2	17 June 2016	08:00:00	12:00:00	4	04:00:00
Project Group Meeting 8	18 June 2016	09:00:00	21:00:00	8	12:00:00
Project Group Meeting 9	19 June 2016	09:00:00	13:00:00	2	04:00:00
Project Group Meeting 10	26 June 2016	14:30:00	17:30:00	2	03:00:00
Reports - Individual and Group	30 June 2016	09:00:00	18:30:00	8	09:30:00
Group Meeting 11(Virtual)	30 June 2016	18:30:00	20:00:00	1	01:30:00
Project Group Meeting 12	1 July 2016	10:00:00	17:00:00	8	07:00:00
					00:00:00
					00:00:00
				97	127:00:00

Appendix B: How to install and run the Twit-Con-Pro Data Visualisation component

Installation

1. Install Node.js from https://nodejs.org/en/download/
2. Getting the Twit-Con-Pro solution:
    a. Option1 – Checkout The Git Project:
       https://github.com/garethstephenson/ELEN7046_Group2_2016.git
    b. Option2 – Download the Git Project from:
       https://github.com/garethstephenson/ELEN7046_Group2_2016

Running the solution

1. For Windows OSes:
    a. Open Command Prompt
2. For Linux and Mac Oses (Including Raspberry-PI):
    a. Open up a Terminal Window locally. Alternatively, open a remote SSH terminal session to the machine hosting Node.js and the solution.
3. Navigate the {solution}\visual folder
4. Execute the following command: "node main.js"
5. The following line of text should appear: "Twit-Con-Pro started on http://:::8081"

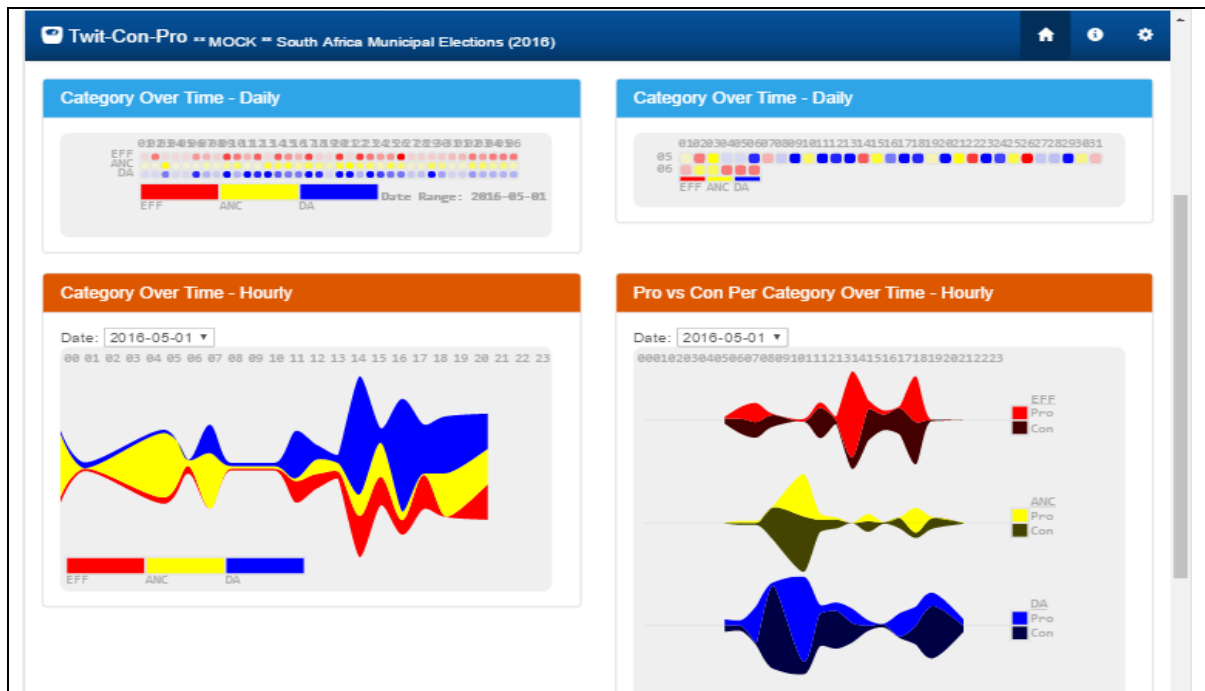## Appendix C: Output Example

Figure 20- Sample View of the rendered dashboard and Graph Views
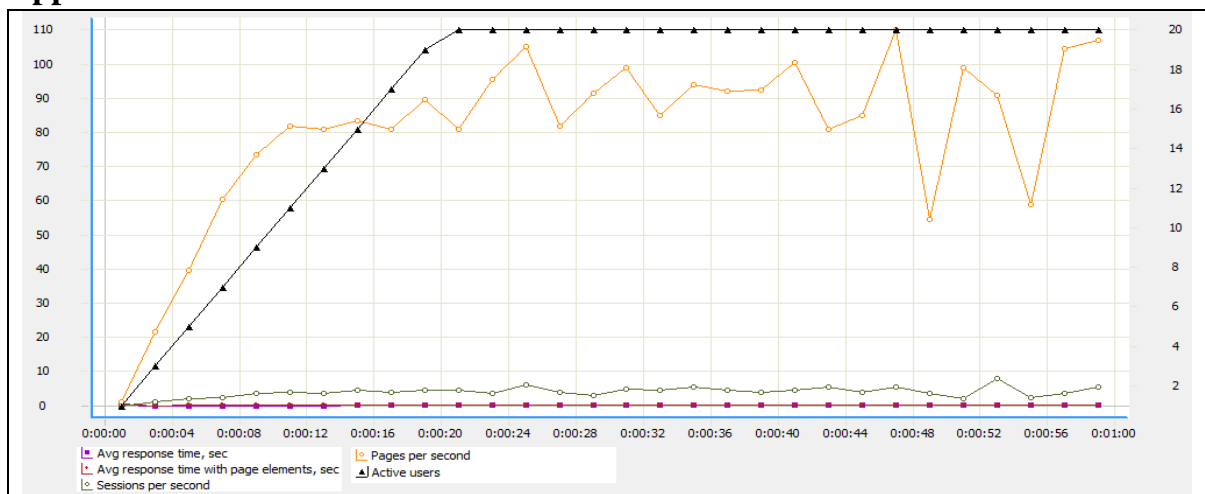
Appendix D: Load test result

Figure 21 - WAPT Load test result using 20 concurrent virtual user connections loading the dashboard and selecting different dates on both the 'Category Over Time - Hourly' and 'Pro vs Con Per Category Over Time - Hourly' Views. The test was performed over a Wi-Fi connection to simulate a degree of network latency. Zero error-rate reported

Summarised Report:

- Successful sessions per second averaging at 3.95 peaking at 4.5
- Successful pages per second averaging at 80.7 peaking at 130
- Successful hits per second averaging at 118 peaking at 136
- Worst average response time with page elements was 0.48 seconds and JSON responses, 0.12 seconds
- Web site remained perfectly accessible during load test with no humanly-notable difference in performance

Appendix E: List of tutorials used

Installing Node.js on a Raspberry-PI: http://thisdavej.com/beginners-guide-to-installing-node-js-on-a-raspberry-pi/

Node.js File System: http://www.tutorialspoint.com/nodejs/nodejs_file_system.htm

Node.js Web Module: http://www.tutorialspoint.com/nodejs/nodejs_web_module.htm

Node.js Express Framework: http://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

Node.js Restful API: http://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm

Node.js – Documentation: https://nodejs.org/dist/latest-v4.x/docs/api/synopsis.html