|  |
| --- |
| Individual Report |
| Big Data Visualization using Commodity Hardware and Open Source Software – Data Visualization Component |
| Course: ELEN7046 – Software Technologies and Techniques |

|  |
| --- |
| Dave Cloete (Student Number 1573016)  6-30-2016 |

Contents

[Abstract 3](#_Toc455166235)

[1. Introduction 4](#_Toc455166236)

[1. Background 4](#_Toc455166237)

[2. Requirements 4](#_Toc455166238)

[2.1. Functional 4](#_Toc455166239)

[2.2. Non-functional 4](#_Toc455166240)

[3. Approach 4](#_Toc455166241)

[4. Challenges 5](#_Toc455166242)

[5. Design Overview 5](#_Toc455166243)

[5.1. Conceptual 5](#_Toc455166244)

[5.2. Technologies Used 6](#_Toc455166245)

[Node.js 6](#_Toc455166246)

[Node.js – Express Framework 6](#_Toc455166247)

[Node.js – File System 6](#_Toc455166248)

[Node.js Twig.js 6](#_Toc455166249)

[Bootstrap 6](#_Toc455166250)

[D3 7](#_Toc455166251)

[Wordcloud2.js 7](#_Toc455166252)

[5.3. Physical Project Structure 7](#_Toc455166253)

[5.4. The Framework 8](#_Toc455166254)

[Dashboard 8](#_Toc455166255)

[Settings 9](#_Toc455166256)

[5.5. Data Models 9](#_Toc455166257)

[Category Color 9](#_Toc455166258)

[Category Count per Day and Category Count per Hour 10](#_Toc455166259)

[Category Summary 10](#_Toc455166260)

[Con-Pro Count per Hour 10](#_Toc455166261)

[Word Count 11](#_Toc455166262)

[5.6. Controllers 11](#_Toc455166263)

[5.7. Component Diagram 12](#_Toc455166264)

[5.8. Graph Views 12](#_Toc455166265)

[.initialise 13](#_Toc455166266)

[.draw 13](#_Toc455166267)

[5.9. Testing 13](#_Toc455166268)

[6. Recommendations 13](#_Toc455166269)

[7. Conclusion 14](#_Toc455166270)

[References 15](#_Toc455166271)

# 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.

|  |
| --- |
|  |

Figure 1- Sample View of the rendered dashboard and Graph Views

# 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.

# 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.

# Requirements

The following is a list of the key requirements.

## 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)

## 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

# Approach

As mentioned in section 2. Background, 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.

To ensure the user experienced a

A web based component was chosen as client-browsers devices would then share the processing required to render charts. Research also shown availability of various open-source web based technologies existed.

A responsive dashboard containing more than one chart was decided on as it would allow cross comparison between charts.

Data file would be placed in a specific directory structure that the Data Visualisation could understand and the Data visualisation framework would simply serve the data from these locations to the data views requesting the information.

Data would be provided to the Data Visualisation component at such a level that little to no aggregation or transformation was to be necessary.

# 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.

# 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.

## 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.

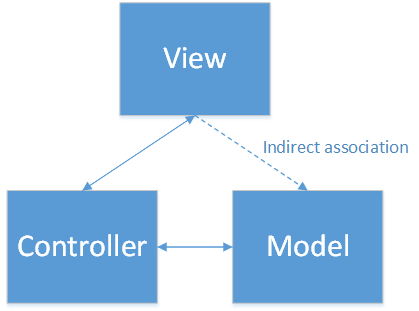


Figure 2 – Depiction of the MVC pattern

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.

## 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 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 (3) 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 (4).

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 (5).

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 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.

## 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 3 – 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.

## 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.

|  |
| --- |
|  |

Figure 4 - 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.

|  |
| --- |
|  |

Figure 5 – 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 6 – 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.

## Data Models

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

### Category Color

|  |
| --- |
|  |

Figure 7 – 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 8 – Field Description

### Category Count per Day and Category Count per Hour

|  |
| --- |
|  |

Figure 9 – 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 10 – Field Description

### Category Summary

|  |
| --- |
|  |

Figure 11 – 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 12 – Field Description

### Con-Pro Count per Hour

|  |
| --- |
|  |

Figure 13 – 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 14 – Field Description

### Word Count

|  |
| --- |
|  |

Figure 15 - Snippet from the WordCount.json Data Model

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

Figure 16 – Field Description

## 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.

|  |
| --- |
|  |

Figure 17 - Snippet from the data.js Controller

Figure 17 is an example of a Data Controller Method.

## Component Diagram

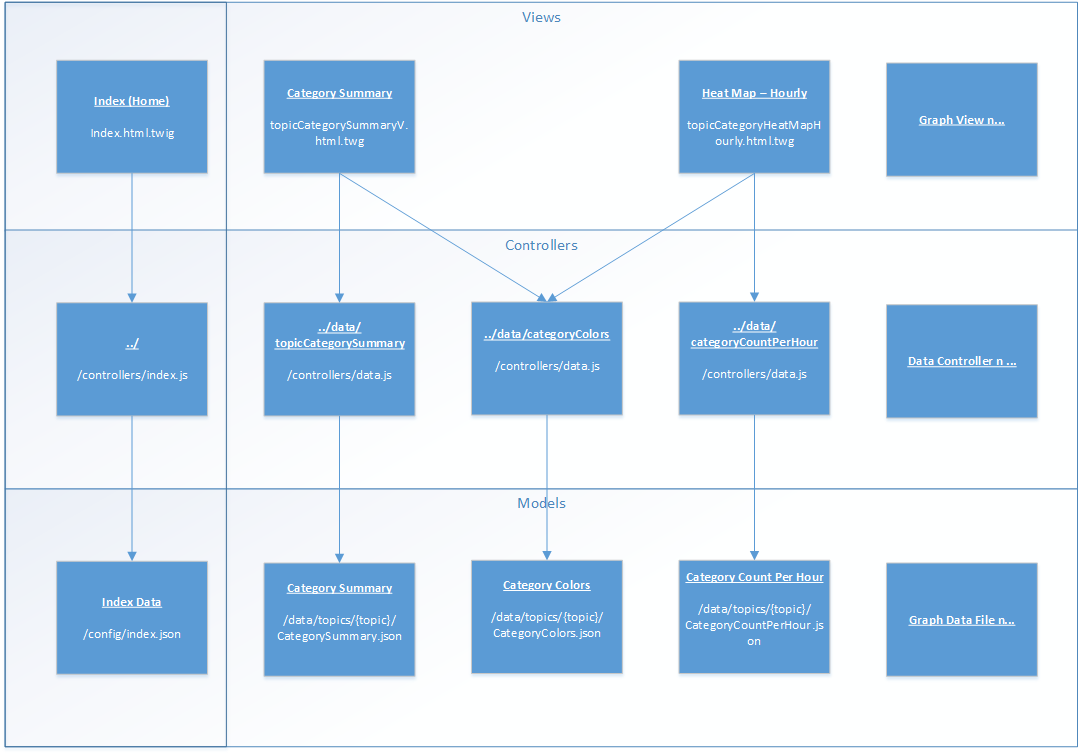


Figure 18 – Data Visualisation Component Diagram

## 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.

|  |
| --- |
|  |

Figure 19 - Snippet from topicCategorySummaryV.html.twig View template showing how the Controller Method is injected into the view

The D3 Queue (6) component was used to retrieve Data Models asynchronously

|  |
| --- |
|  |

Figure 20 - 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.

## 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.

# 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 JQueryUI 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.

# Conclusion

todo

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

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

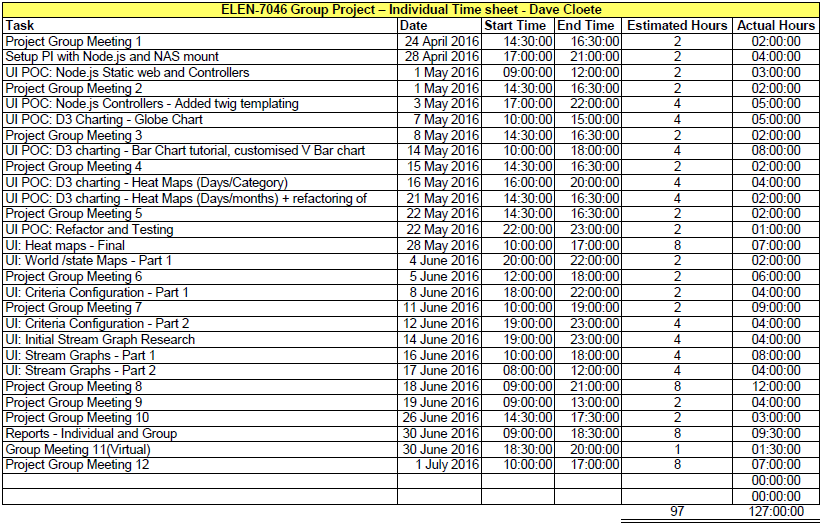
5. **D3.** Overview. *D3.* [Online] https://d3js.org/.

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

7. —. Let’s Make a Bar Chart. *Mike Bostock.* [Online] https://bost.ocks.org/mike/bar/.

8. **NPM.** Packages - twig. *NPM.* [Online] https://www.npmjs.com/package/twig.

# Appendix A – Timesheets



# Appendix B – Installation Instruction