**CHAPTER 4**

**RESULT AND ANALYSIS**

This chapter provides the result and analysis of this project which includes the system flow, system development flow and system user interface (UI) explanation. This chapter will also focus on the process and the challenges faced during the development of the system.

**4.1 Introduction**

The main functionality for this system is to convert the election data into a hexagon tile grid map visualization. The election data were manually stored into a JSON file to make the retrieval of the data as smooth as possible.

The system is developed by using a combination of HTML, Cascading Stylesheet (CSS), Javascript (JS) and Data Driven Document (D3.js) together with D3.js hexagon plugin, hexbin. The HTML and CSS were used to setup the foundation of the User Interface (UI) of the system while JS and D3.js were used to implement the algorithm flow of the system such as creating and populating the hexagon based on the coordinate data stored in JSON file. The hexbin plugin of D3.js were used to help creating the hexagon tile for a better coordination. All of the output seen in the HTML document are in Simple Vector Graphic (SVG) format produced by D3.js.

**4.2 System Modelling**

This project consists of two types of files, namely the HTML file and the JSON data file. The HTML file contains the layout of the system UI and the logic behind the hexagon tile grid map. The logic uses D3.js, which is a Javascript library used in the same HTML file through the <script> tag. JSON files contains the data that are needed to process and populate into the system UI. The file structure is shown in Figure 4.2:

|  |
| --- |
| **Figure 4.1** File structure of the system |

Based on Figure 4.1, the files that are in the “json” folder contains the JSON data file that stores the data needed in order to run the system. The file, setting.json stores all the setting data such as the height and width of the SVG element, padding of the SVG element, radius of the hexagon, tooltip position and colors for the hexagon based on political party.

The parlimen-v1.html is the HTML document for the first version of the system. The parlimen-v2.html is the HTML document for the latest version of the system. This chapter explains about the latest version of the system which is parlimen-v2.html.

Files in “lib” folder contain all the library and dependencies that are used in the system. The file d3.min.js is the core library for D3.js, d3-hexbin.min.js contains the extended plugin of d3.js used to draw the hexagon and jquery-3.3.1.min.js contains the core library for jquery that will be used in the system.

The HTML files is the main file for the system that contains the HTML document for the UI and the Javascript for logical operation of the system. The structure of the file is shown in Figure 4.2:

|  |
| --- |
| **Figure 4.2** Main file coding structure, parlimen-v2.html |

Based on Figure 4.2, there are two main sections in the file. These sections are the HTML document section and Javascript section. The HTML document section contains the division that shows the data visualization. The Javascript section contains the import section and the script section. The import section is where all dependencies and libraries such as Jquery, D3.js and hexbin plugin are imported into the HTML document and the script section contains the logical part of the whole system.

**4.2.1 Single Hexagon Tile Setup**

This project uses the hexbin plugin for D3.js to draw the hexagon tile. Its ability to easily create a single hexagon tile is one of the reasons why this plugin was chosen in the first place. Hexbin plugin will produce a SVG path based on the developer’s configuration. The SVG path will eventually create a hexagon tile. The plugin generates the hexagon from the perspective of a circle. Therefore, the main parameter that the developer had to include is the radius of the hexagon tile and it is considered as the size of the hexagon. After that, the developer can plot the hexagon anywhere inside the SVG plain. Figure 4.4 shows the code for single hexagon tile setup:

|  |
| --- |
| **Figure 4.3** Coding for a single hexagon |

Figure 4.3 shows the code to draw a single hexagon using the D3.js library and hexbin plugin. In variable declaration section (from line 21 until 27), *coordinate* variable stores a list of coordinates (in this case, one coordinate) that are used to plot the hexagon. The variable *svg* stores the instance of the <svg> in the <div> element with id of “chart” and variable *hexbin* stores the instance of the hexbin plugin with the hexagon radius being declared as “10”. This means that the size of the hexagon is drawn based on the circle radius of 10 pixels. However, the size of hexagon is adjustable according to the development need. The developer can increase the radius to enlarge the size of hexagon and vice versa.

In the process section (from line 29 until 37), the real operation of the hexagon drawing takes place. Firstly, line 30 shows a D3.js selection function *.selectAll()*, which selects all of the occurrence of <g> tag in the HTML document during execution. On line 31 and 32, for each data stored in the *coordinate* variable, the system will create a new instance of <path> tag inside the <g> tag. On line 33 until 35, the attribute of the <path> tag are initialized by using the *.attr()* function.

Most of the function in D3.js requires 2 parameters, name of the attribute and its value. The <path> tag requires the developer to include one main attribute, the *d* attribute. The *d* is used to draw a path by using the <path> tag in SVG plane (inside <svg> tag). The value for *d* attribute is a set of strings that determine the command for the drawing of the <path>. The value included for *d* is the coordinate command that includes the coordinate to which the hexagon will be plotted (as stored in the *coordinate* variable) and the shape drawing command the hexbin plugin generates by using the *.hexagon()* function. On line 36 and 37, the styling of the hexagon are stated using the *.style()* function. In this case, the style attribute that are being initialized is *stroke,* the border line color of the hexagon and *fill,* the color inside the hexagon. Developer can choose any color for the *stroke* and the *fill* attribute. The function will produce the output as shown in Figure 4.4:

|  |
| --- |
| **Figure 4.4** Single hexagon output |

**4.2.2 JSON Data File Setup**

[3 data file, settings.json, parlimen.json & pilihanraya.json, problem with reading data in json file using d3.js]

The first data file is settings.json. This file contains the settings required for the layout of the user interface. Figure 4.7 shows the structure of the data file. JSON file can be created using simple notepad.

**4.2.3 Hexagon Tile Grid Map Plotting**

This project implements the pointy-top hexagon because the hexbin plugin are not developed with a flat-top hexagon in mind. However, by using hexbin plugin, flat-top hexagon can still be created nonetheless just by transforming (rotate) a single hexagon, but there are problems related with the rotation of the hexagon. The problems are explained in detail in the discussion section.

|  |
| --- |
| **Figure 4.6** Code for hexagon plotting |

**4.2.4 Mouse On Hover Tooltip Setup**

[tooltip will show parliament code & name]

Hover over as one of the event listeners.

Once this event listener is triggered, a tooltip pops up

The content depends on the developer needs. However, in this system, the content of the tooltip is limited to parliament information such as parliament code and name.

**4.2.5 Mouse On Click Popup Setup**

[popup will show the election data]

**4.2.6 Hexagon Tile Colour Setup**

**4.2.7 State border Setup**

**4.3 User Interface**

**4.3.1 First Level information**

**4.3.2 Second Level Information**

tooltip dan event handling hover over shows parliament number and name

**4.3.3 Third Level Information**

**4.4 Evaluation**

**4.5 Deployment**

**4.6 Discussion**

Even though flat-top hexagon can still be achieved by using a transformation method (rotation), the coordinate system will not be the same and this will lead to a randomly generated hexagon even though the developer uses the same coordinate for each of the hexagon. For example, a pointy-top hexagon coordinate is at (0, 0), but when it is rotated 90°, the coordinate that were registered will still be the same which is (0, 0), but the actual coordinate that the were shown by the browser is (-10, 10) which in this case is not the same as the expected output. The expected output is a rotated hexagon tile with a same coordinate.

**4.7 Summary**