**SENG2021 – Software Engineering Workshop 2B: Design Report 1 Version: 2.0**

Authors: Group 10 - Hariharen Veerakumar, Yuwei Chen, Ahsan Rahman, Saishav Agarwal

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Scope:**

The team will construct a web page that will allow users to view and analyse each Australian Political Parties’ information. The infographic displayed will have two main functions. The first will be a graphical repesentation of the seating numbers of each party. Furthermore, there will be acess to each of the party's policies from this infographic.

This information is envisaged to be displayed via one dynamic infographic. Subsequently, this dynamic, accessible yet simple layout heightens the user’s experience. This information will be mainly catered for people who are less politically-minded, and are looking for quick summarised information to get a concise overview of Australia's major parties.

Use Cases:

![A description...](data:None;base64,)

System Functions:

At the initial level of the website, the information will be displayed via an infographic. All the political parties will be displayed in free moving bubbles with each individual bubble containing the parties’ logo. The size of the bubble will be indicative of the proportion of seats that are owned within each of the \*\*\*Houses\*\*\*, the House of Representatives and the Senate. Buttons will be positioned to allow a dynamic change between the two options.

\*\*\*Furthermore, the display of policies will be in a less graphical manner, but is intended to be more concise and summarised. This

Example

A user opens the website on the 12/09/2013 and is faced with all the political party logos floating around the screen. In this case, the Liberal party, holding the most number of seats in the House of Representatives, will have the largest bubble, followed by the size of Labor's circle. The circles are in a randomised order, and randomly generated upon use.

The initial view will also have two buttons at the bottom, labelled “Senate” and “House of Representatives” that the user can click to view the statistics of the seat domination of each party between the different \*\*\*houses\*\*\*. By default, the initial view displays the House of Representatives. When one particular view is showing, this button is disabled.

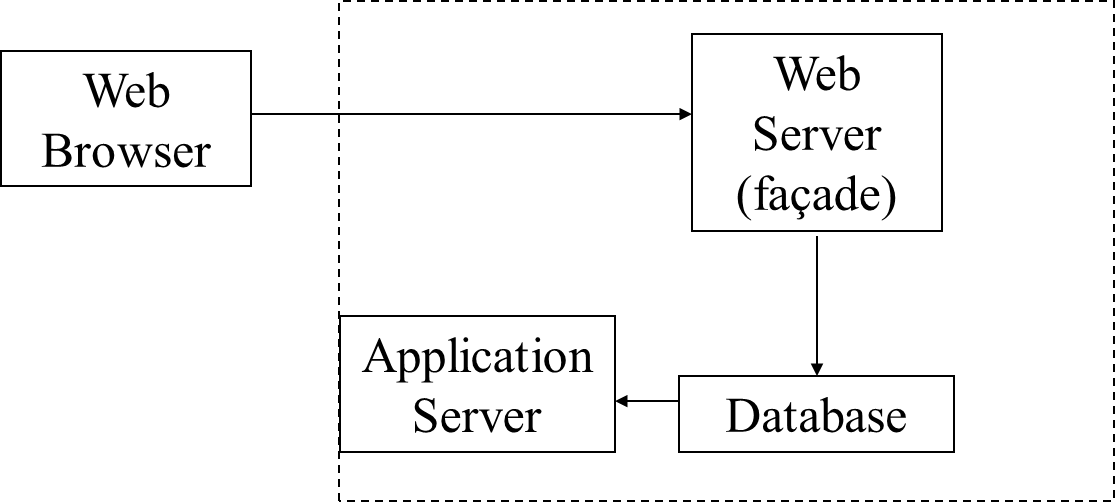
The user wants to learn about the Australian Labor Party, and thus the user clicks the Australian Labor Party bubble. A dynamic translucent browser pops up (within the same page) over the screen. This pop up displays\*\*\*.

The pop-up windows at the final level that displays the information for \*\*\* of a specific political party, will contain links to external sources for users to access further reading material. The user, at any level can click the bubble again to return to the preceding level.

A user opens the website and is faced with all the political party logos floating around the screen, the user wants to learn about the Australian Labor Party, and thus the user clicks the Australian Labor Party bubble. A dynamic translucent browser pops up (within the same page) over the screen. This pop up displays the specific categories they can learn about; a bubble for ‘Policies’, ‘Marginal Seats’, ‘Stronghold Areas’. If the user wants to learn about the policies, they select the Policies bubble which again causes another pop up window over the bubble they selected. The newly popped up window will contain bubbles with all the specific policies. Moreover, if the user wanted to learn about ‘Carbon Tax’, they would click the 'Carbon Tax' bubble, which will consequently open a new pop up window over the bubble that displays all the relevant information.

The user, at any level can click the bubble again to return to the preceding level. For example, if the user selected ‘Policies’, which in turn produced a number of different options but then wanted to view ‘Marginal Seats’, they would click the ‘Policies’ bubble, which can be still seen translucently behind the new window, to return to that level. Windows can also be closed, returning us to previous level, by closing the window using an integrated close button. For each window that has popped up, while it is active, no other bubble is available to select on the web page. The pop up window must be closed before a user can select another category. All windows will be scrollable windows, in order to fit information within the one website page. The pop-up windows at the final level that displays the information for a specific topic of a specific political party, will contain links to external sources for users to access further reading material.

**Software Architecture**



The design will be based around the client-server Web architecture. This was deemed the most convenient for our purpose, since there will be a variety of back-end needs required to produce the desired result, especially considering its dynamic nature.

We have decided to use a Model-View-Controller proxy (MVC) pattern to help organise the various components of the system, such that each component can be dedicated to a simple task in itself.

Components

The main necessary component, given a client, is a variety of server components and systems within a computer itself. Provided this server can run indefinitely, the design will work such that the client can access the information at any time necessary. The client view is generated as a web application for dynamic use by the user. The client will also be offered options to act as a controller. Interacting with these will send requests to the server, acting as the “model” which, in turn, will change the “view” that is displayed by the client.

A web server is required to act as the “model” in our MVC architecture. It essentially creates the website and sends the information to the client. Various templates, and other data, will be accessed by this server and transmitted to the client upon request from the “controller”. The web server will retrieve information by contacting other servers, such as from www.alp.gov.au (Australian Labor Party Official Site) to collect details regarding their particular policies, for example. Considering some of this information can change extremely readily, we will have to collect this dynamically.

Relationships between components, and deployment

The web server will require applications for their respective tasks, mainly concerning the sending and receiving of complex data to and fro other components. In particular, the web server will have to retrieve data from the DBMS, and receive requests and send replies back to the client. Similarly, the application server will also send and receive data to different components – namely, for the receiving of requests from the web server for data and the sending of data back and stored into the DBMS.

Lastly, the type of browsers to use was chosen based on a universal scoring system used to judge the rendering ability of HTML5 and CSS3 in particular. From experience with all browsers throughout time, we decided that the fairest minimal score, considering the available upgrades for each browser, would be 300. This was also a key determining factor for which OS compatibilities we will be catering for, as seen in the next section.

The browsers chosen include Internet Explorer 10 (scoring 322), Chrome 7 (308), Firefox 6 (333), Safari 5.1 (319) and Opera 11.10 (301).

**Implementation Considerations:**

Choice of Technology:

We have decided to use Ruby and Ruby on Rails for the back-end of our web system i.e. for server functions. Ruby is been proven to be a very natural language for programmers, which means that it is easy to pick up. It is object-oriented and also supports functional aspects of programming, making it robust. Good programming involves writing minimalistic code to perform tasks, and Ruby, as compared to Java and C, requires less amount of code to be written, making it more efficient. Besides, Ruby is not a restricted language. Users have the authority to customize it by adding or removing any methods within the built-in classes, as per their needs and requirements.

Ruby on Rails is very useful in writing web applications. It allows the server to fetch and store real time information. It is easy to navigate and makes writing web apps quick and simple.

For both the client view and controller, we will be using JavaScript. JavaScript has syntax that is similar to C making it easy to code. It allows users to embed dynamic functionality on web pages. There are other alternatives to JavaScript like VBScript, but the reason for choosing JavaScript is that it is fully compatible with most of the popular browsers. It also allows dynamic retrieval of information. JavaScript has been successfully used for the last decade with most HTML web pages, which gives us good reason for making it our choice.

Relating choice to Components:

We have decided on using the Client-Server architecture for this project. This is because the client-server architecture makes recovery of data possible, in case of data loss. Also, changes can be easily made and new resources can be allocated to the server. It is secure because, the server transmits only important data to the client, when requested. One big advantage of using this architecture is caching. Data that is frequently/likely to be used can be stored on the server end, and retrieved whenever necessary. This saves the hassle of retrieving data from another web page, since that would be much more time consuming.

Choice of Platform:

As far as the platform is concerned, we will keep it to a basic computer OS to make it accessible to maximum number of people. The webpage will work on the three most commonly used operating systems, which are Windows, OSX and Linux. However the minimum requirement for Windows is Windows XP and for Mac is OSX 10.5. The web page might run on previous OS versions but will not be completely compatible as some features are only added on the later updates of the browsers compatible with the OS.