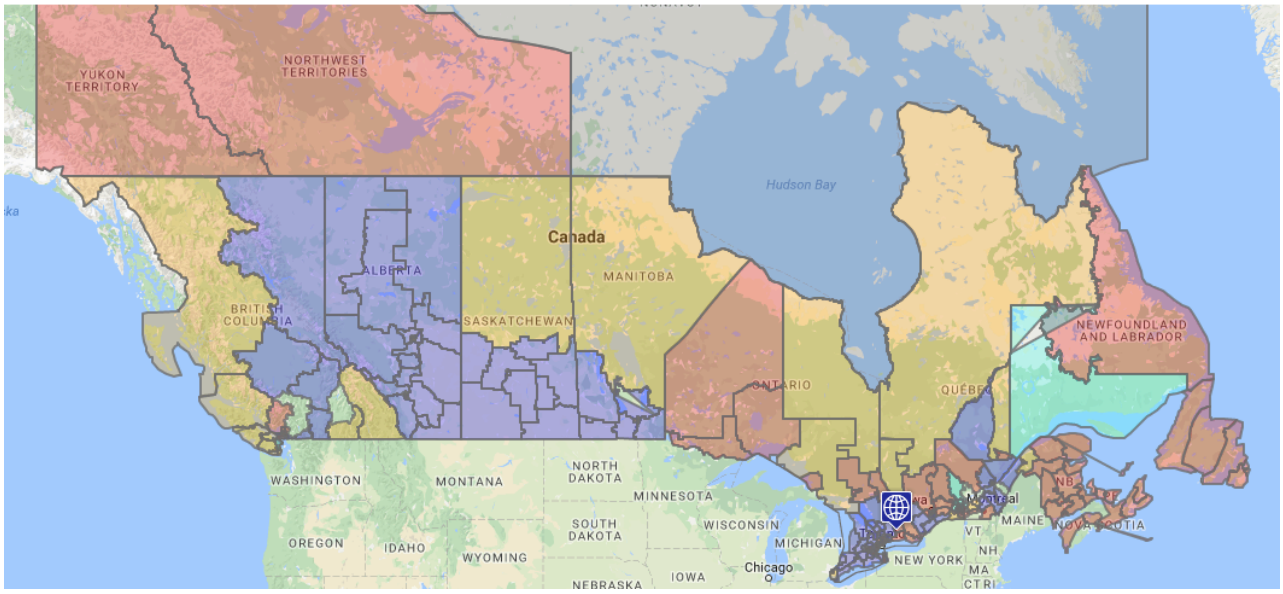
UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

SOFE 4790U: Distributed Systems

Electoral District Heat Mapping

Nick Mancini – 100517944



Submitted to the University of Ontario Institute of Technology
Faculty of Engineering and Applied Science
Department of Computer, Software, and Electrical Engineering

28 November, 2016

Table of Contents

Table of Figures

Abstract

Canada is a geographically vast and politically diverse nation, and as such it is often difficult to obtain a sense of the political alignments in this nation's various regions. For example, some regions may politically oppose the ruling party, and for a person living in one of these regions, it may be difficult to perceive how such an apparently unpopular party managed to be elected in the first place. Furthermore, given the relatively unbalanced representation of various electoral ridings, some regions may achieve more representation per citizen than other, more populous regions, further giving argument to the notion that an elected party seems to be only supported by a minority of people. When it comes time for an election again, candidates often travel to ridings they find important or tantamount to their victory, though this is sometimes difficult to judge from polls alone.

To help accommodate this problem, a heat-mapping web application that clearly indicates all political ridings across Canada as well as their present political affiliation has been developed. The goal of this system is to give individuals a sense of current political support for a given party. For undecided voters, this tool demonstrates general allegiance in their riding (or even region) and assist them in deciding based on their own views of how their riding is faring under the administration the current representing party. For candidates running for office either at a federal election or in a regional by-election, this system shows the political preferences of various ridings. The system employs the Google Maps API as well as data obtained via the Represent Civic Information API, hosted by Open North to create a politically coloured heat map of Canada, subdivided by ridings. When a riding is clicked, information about that riding, such as affiliation, member of parliament, and population as of the last census will be displayed in a popup message. This map retrieves data from an up-to-date online API. Every time there is a riding change as the result of a federal or by-election the data can be re-downloaded and the map will be refreshed. The implementation of this system has made the Canadian political environment easier for all individuals who make use of it.

Project Statement

This project is intended to deal with the confusion of viewing Canada's political landscape by taking up to date data and turning it into a graphical format that is easy to understand. Using a Node.js server, and Represent Civic Information API, the system gathers the electoral districts mappings across Canada as well as the representing party, and supplies them to an AngularJS web client in JSON format. Once the information is received, the client interprets the shape instructions and political party provided, and, using the Google Maps API, iteratively populates a map of

Canada with coloured electoral districts, whose colour will be determined by the political party administering said district (See Figure 3, under *Usage Instructions*). After the render is complete, the map is shown to a user on an interactive single-page website (with URL http://localhost:3001), with a "Refresh" button visible below the map. The system will continuously maintain contact with the Node.js server, and when the refresh button is clicked, the previously retrieved information from the server is deleted, and a new set is downloaded, thus providing the most up-to-date map at the time of refreshing. For example, if a different political party wins in a by-election, and the refresh button is clicked, the colour of that riding will change to match the new party. Clicking on any riding will query the server with coordinates of the click spot, and, upon reply, will display a popup at the location of the click indicating the name of the riding, the representing party, the member of parliament and their e-mail address, a link to more information about the member of parliament, and the population of the riding as of the most recent census.

This application is a two-part self-contained application with a client and server configuration. The server, acts as a go-between for the client and API, as it accesses Open North's data on the client's behalf, and assembles the data in a desired format, before replying to the client's request with the data. All that is required is an internet connection so that the Node.js server can retrieve electoral data from Open North. After the first download, the electoral data is stored in cache, and the server is not accessed again until the refresh button is clicked, or an electoral district on the map is clicked. This not only makes subsequent accesses to the client faster in the future, but also, in the event of no internet connection, allows the application to show the user the last updated electoral map. If there is no connection, an error is displayed in the JavaScript console. In the potential circumstance where the cache is empty, and the server cannot be reached, the map will simply remain unpopulated, and error messages will be displayed in the JavaScript console.

Components

Frontend

1. AngularJS Framework (v1.5.8)
2. ng-map plugin for AngularJS (v1.17.94)
3. Google Maps V3 API
4. HTML5 and CSS3
5. Twitter Bootstrap Frontend Framework (v3.3.7)
6. Python SimpleHTTPServer (v2.7.12)

    a. NOTE: For easy deployment, Python is recommended, however, an HTTP server of any kind, such as Apache, can be substituted.
7. EMCAScript5-compatible web browser (Google Chrome 54 was used during development)

Middleware

1. Node.js (v6.3.1) and dependencies
2. Express.js (v4.14.0) and dependencies
3. Represent Civic Data API Library for Node.js (v0.1.0) and dependencies

Backend

1. Open North's Represent Civic Data API

Architecture

The system is set up in the form of a three-tiered architecture, with a client, middleware server, and API server. When the middleware and client servers are started, and the webpage of the client is loaded, the client checks for cached map data. If none is found, it issues a POST request to the middleware server, asking for the shapes and parties of all electoral ridings in Canada. The middleware server then queries the API server for a list of all federal electoral districts in Canada. This information is returned as a JSON array, which the server then iterates through, calling the API server with a request for shape information each of the 318 available districts (There are actually 338 districts, however the API appears to be missing 20). When the shape data for a district is returned, the middleware server then queries the API server for that district's party (which is in a separate location from the shape data). Once that data is retrieved, the shape data and party are amalgamated into a single JavaScript object, and pushed to an array. The server then repeats the process for the next district.

Once all the districts have been queried, and the data has all been pushed into the array, the middleware server replies to the client's POST request with the array. Upon receiving this reply, the client places the received data into a local scope object, and a copy is stored in the browser cache. Next, the client calls a function to draw the shapes on the map with this object as an argument. This method formats and passes the data into an object array connected to the Google Map on the HTML page, and uses an if/else block to determine the shape colour from the party. Once all the data has been passed into this object, the map draws the shapes and makes them visible to the user through the HTML page.

At the same time as this information downloading and processing is happening, the client asks for permission to use, and then discerns the client's location, placing a marker on the map at the deduced location.

After all this downloading is done the application is now ready for use. The user can navigate the map and view nearly every electoral district in Canada (20 of the 338 are missing from the API for unknown reasons). If a user wishes to know more about a district, they simply need to click on the district, and the client will send a new POST request to the middleware server, with the latitude and longitude of the click location in the body of the request, and open an information marker at the click location.

Upon receiving this request, the middleware server once again calls the Represent API with the coordinates it was provided, and retrieves the external identifier for the district that contains the coordinates (i.e., if the coordinates were located inside this shape, what riding would that be?) as well as its official name and stores them locally. Now in possession of this external identifier, the server will send a new request to the API, asking for the population of the district as of the last census. Once it has this data, it once again stores the values locally, and send a third and final request to the API asking for the political party, the name of the member of parliament, their e-mail address, and a URL to Parliament's official site for more information on the individual. Once it has this data, it checks the data to ensure that none of the values are undefined. If any are, the district is assumed to be vacant, an object is constructed containing the name and population of the district (which always exist), as well as fields for Party, MP, MP E-mail address, and Parliament URL which are all given values of either "Vacant" or "N/A". If none of the values are undefined, an object is instead constructed containing all the fields above, but instead populated with the actual retrieved data:

```
replyPayload = {
                    name: localData.name,
                    mp: data.objects[0].name,
                    email: data.objects[0].email,
                    moreInfo: data.objects[0].url,
                    party: data.objects[0].party_name,
                    population: localPop
                };
```

With this data populated, the middleware server sends it to the client as the reply to the client's original POST request. Upon receiving the reply, the client places the object in a local scope variable which, is utilized by the map as the body of the information marker.

Due to the asynchronous nature of the request, the popup may appear on the map before the information has been retrieved, in which case, the popup will contain labels or the data, but no actual data (except for population, which is simply initialized to a default value of -1), however, once the data is retrieved, the popup will appear, showing all the information that the server has replied with (see Figure 1).

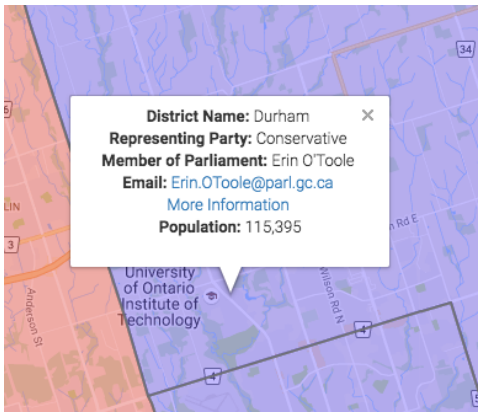


Figure 1: Example Popup Window

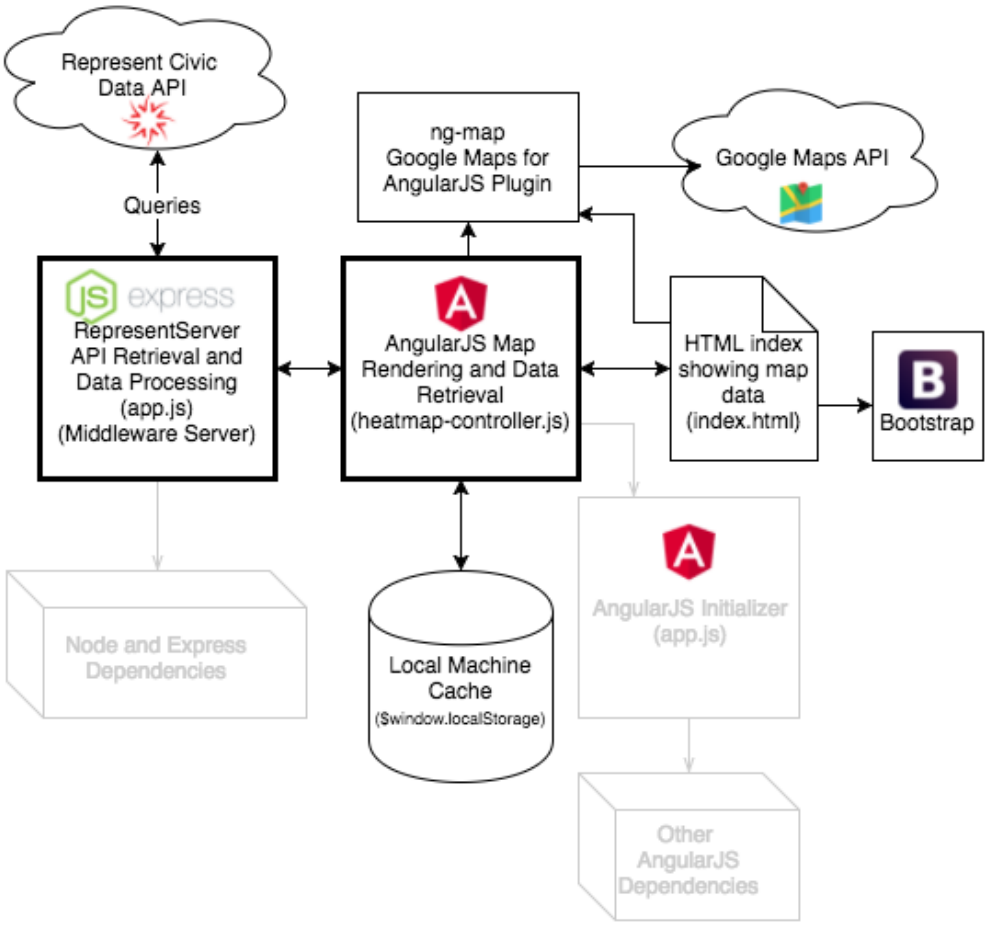


Figure 2: Heat Map Overall Architecture (trivial components are light gray, main components are black)

Usage Instructions

This project is a web application, and is therefore cross-platform and compatible with any operating system.

1. If you have not already done so, please ensure you have Node.js and Python 2.7 installed:
    a. https://nodejs.org/en/download/
    b. https://www.python.org/downloads/

2. Open a Terminal or Command Window in the Project Folder

3. If you are running a Unix-Like Operating System (i.e., macOS, Linux, etc.) enter the following command, and then go to step 5:
    a. `./ElectionMap.sh`

4. If you are running a non-Unix-like OS (such as Microsoft Windows) the servers will need to be started manually.
    a. Open two terminal windows.
    b. In the first, enter this command:
        i. `node RepresentServer\bin\www #Starts server on port 3000`
    c. In the other, enter these two commands in the following order:
        i. `cd RepresentClient\app #Changes to client main directory`
        ii. `python –m SimpleHTTPServer 3001 #Load client on port 3001`
    d. Launch the web browser of your choice, and navigate to http://localhost:3001

5. The web application will begin loading, for more verbose information, open the developer tools console (in Google Chrome, you can hit F12 and select the "Console" tab). Please wait for the information to be downloaded from the remote server. The data totals approximately 7MB in size, however, it is retrieved iteratively, making the download process take up to 30 or 40 seconds. Once the data is downloaded, it will be printed to the developer console in JSON format, please allow another 10-20 seconds for the map to draw. During this time, the map will appear to be frozen. **Once the map is populated, the application is ready to use, and the downloaded data is cached, making successive accesses to the site faster.**

6. The map should be colour-coded by electoral district, with a marker on your machine's apparent location. The colour coding is as follows:

Party	Colour
Liberal	Red
Conservative	Blue
NDP	Orange
Bloc Quebecois	Cyan
Green Party	Green
Independent	Gray

Party	Colour
Vacant/Other	Magenta

Figure 3: Electoral Heat Mapping Colour Scheme

7. To get more information on a district, click anywhere on it. A popup will appear at the click point. Allow 4-5 seconds for the data to download. Once downloaded, the popup will list the district name, its representing party, the member of parliament, the MP's e-mail address, a link to more information on the MP, and the population of the district as of the last census.

8. To get new data, click the green refresh button under the map. This will clear the application's cache, and re-download the data from the API.

9. When you are done with the application, it can be terminated by running the EndProgram.sh script (read disclaimer at the bottom of this section before doing so). If you are on Windows/Non-Unix-Like, simply type Ctrl+C (^C) into the two terminal windows created in step 4, this will manually terminate the node and python webservers.

DISCLAIMER: Running EndProgram.sh will issue a general **'killall'** command to any processes with the name Python or Node. If you are running and important applications with either of these programs, it is strongly advised to manually terminate the heat map processes from your system monitor/task manager.

Conclusion

This heat mapping system will be able to represent Canada's political landscape in a way that is easy to perceive and understand, and hopefully provide voters and candidates alike with a solution that helps them decide on strategies and voting methods with every coming election in Canada for the foreseeable future. It is hoped that the benefits of such a system, as outlined in this report will be viewed as a beneficial and worthwhile tool for all who use it. Thank you for taking the time to read this report and try the Electoral Heat Mapping Application.