

Visualizing Technology Use by Age and Region

Project Documentation

The team must provide a document that is an update of the former, documenting what was done to implement the client-side of the application. What functionalities were implemented? What were added that were not planned? What functionalities were removed? Such information should be provided in this document.

Our functionalities have always been for the express purpose of providing useful data visualization. As we integrated geoJSON for displaying our visualizations by location, we realized that, while not infeasible for our codebase, properly displaying both by location and allowing the visualization of multiple different custom searches would require significant unplanned functionality beyond the scope of our project.

We have thus decided to significantly limit the breadth of possible searches so as to preserve the core function of comparing data province-by-province. Additionally, for time, we implemented a simple coloration of the map rather than charts located on it. We feel the charts had minimal impact on proving our ability to code what was learned in this class, while implementing them would've taken considerable development time.

In the process of implementing client-side functionality to create a custom visualization(aka, a "search", or graphing), we discovered that additional work was required to properly handle ages, as we want the user to experience parity between the datasets we're displaying together. With this work done, we changed our solution for multiple locations to match, which simplified our code.

We also took on board feedback from the previous iteration, which noted large inefficiencies in our code related to checking query strings. With the above change in how we handled multiple values and having learned client-side code in class, tight control on the client-side of possible inputs has rendered our previously existing validation functions completely unnecessary, and it has been removed. We note that our dataset is only a few thousand entries, and so outside of this, no special effort was paid to discover other efficiency opportunities.

While implementing the leaflet.js map we discovered it has support for geoJSON files. These files contain polygons meant to encode geographical locations. We found some Canadian geoJSON files at <https://github.com/johan/world.geo.json/pull/29>. With the combination files created for Atlantic region and Prairie provinces the geoJSON files totaled over 2MB. <https://www.npmjs.com/package/mapshaper> was used to compress the geoJSON to 200 KB without loss of fidelity.

Our implementation for exporting changed slightly from our previous iteration. Our initial implementation has us generating the tsv file server side and sending it with an Express.js function called 'download'. This failed as most modern browsers do not support this function. We opted to generate the tsv representation on the server side and send it as text. When the client receives the text a Blob is used to construct a url. A hyperlink html object is created targeting the url, and added to the document. This object then has .click() called on it starting the download, then is removed from the document.

```
const url = window.URL.createObjectURL(
  new Blob([response], { type: "text/plain" }))
const a = document.createElement("a")
a.style.display = 'none'
a.href = url;
a.download = "usage_export.tsv"
document.body.appendChild(a)
a.click()
window.URL.revokeObjectURL(url)
```

Views

The document must have a section named Views where you document your views and what you can do with them (methods). Why are they necessary for your application?

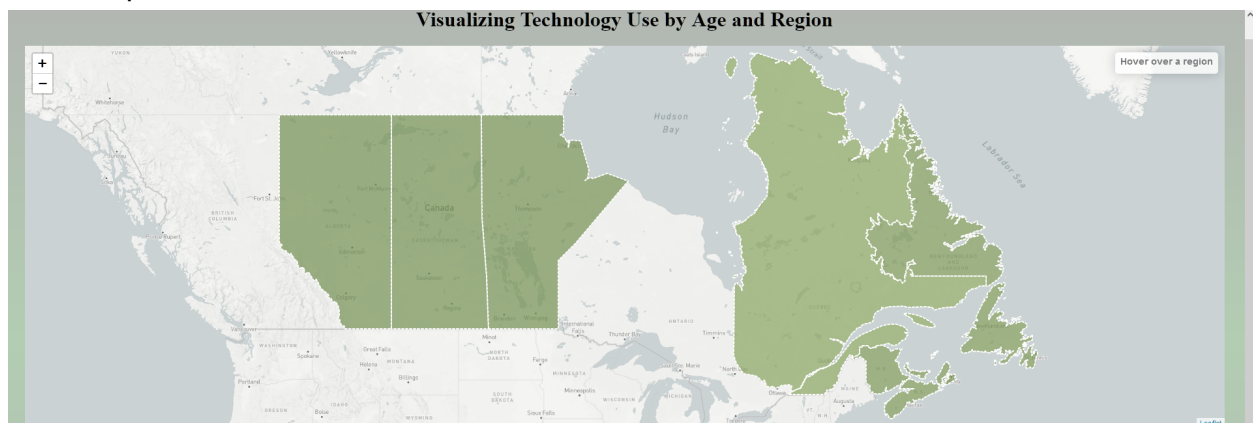
We have two views. We have our leaflet map, which serves as a display for the particular data requested by the user and breaks up that data by location. Without this view, we would not be providing the user with any ability to explore the data and tease out the differences that are quite difficult to see from the .csv's provided by the Government of Canada.

The map's javascript logic is separated in a file named map.js. Our main use for the map is to get our data from the server and call drawUsages() or drawResponses() to have it drawn on the map. A simple interface for drawing and clearing specific regions paintRegion() and clearRegion().

Coloring logic is handled internally by map.js using data received from the server. 3 color modes are supported for the map. One is used for each individual dataset and the last form when they are both enabled.

The overlay is also managed internally by map.js. It's role is to provide an interactive feel for the user while they pan around the map while providing useful information. There are multiple states the overlay has to monitor to display accurate information to the user. The majority of its implementation is based around checking the state of the front end and displaying the proper message for the situation/

Here's a picture of the view.



Our second view is the data selection, where the user configures the parameters of their search to the server for display on the previously mentioned map. When a Region is turned on or off the map's state will be updated to represent the change of locations. The remaining options cause no updates when they are changed. When the user presses the Start Graphing button, the inputs are read and the query is constructed.

Here's the picture of that (note that, with both datasets are deselected, none of our dataset-specific options are visible which makes this take up less space in our report):

Select dataset(s) For information on these dataset please see: [Opinions on Technology](#), [Usage of technology by Age](#)

☐ Opinions on Technology ☐ Use of Technology by Age

Select age range to include.

15-24 ▼

Select at least one Location/Region, but as many as you like!

<input type="checkbox"/> Canada	<input type="checkbox"/> Atlantic Provinces	<input type="checkbox"/> Quebec	<input type="checkbox"/> Ontario
<input type="checkbox"/> Prairie Provinces	<input type="checkbox"/> British Columbia		

Elements and Animations

The document should list and explain the elements used in assembling the client-side. Describing the animation effects used is also important in this part of the report.

In terms of elements, we require significant amounts of user input, and so the majority of our elements are input elements.

Outside of the leaflet map, all elements are in fact some type of input or selection, which we use to build a custom object (essentially a dictionary) that is passed to our backend, further modified, and sent on to MongoDB as a search.

We have dropdown selections for age, which is a shared value in both datasets, and for sex, question, and answer, which belongs to and is handled by Responses(The Opinions on the use of technology dataset), as well as a dropdown selection for each possible statistic in the Usages (Usage of technology) dataset. If a dataset is not selected, all of their related options fade out, and become visible again if the dataset is later selected.

We have two collections of checkboxes, one for selecting what dataset(s) you would like to visualize, and one for selecting which location(s) you would like included. Due to the limitations of the Usage of technology dataset, we limit the locations displayed when it is selected, and their disappearance is a fade-out like for our dropdowns. Only locations exclusive to its sister dataset fade out, and each is thus coded individually.