Ricardo Oliveira

Free and Open Source Software for Geospatial Applications – Term Project

Creating an Web-Map based on FOSS4G Stack

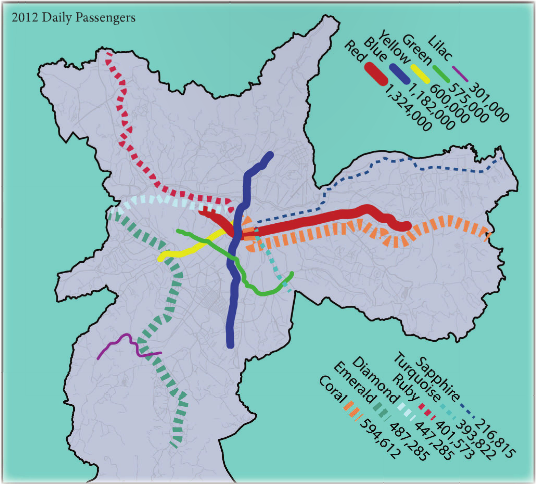
This project aims to provide a concise narrative of how it is possible to build a web-map application using only free and open source alternatives. For this project the following software were used:

* QGIS[[1]](#footnote-1)
* PostgreSQL[[2]](#footnote-2)
* PostGIS[[3]](#footnote-3)
* Leaflet[[4]](#footnote-4)
* Brackets[[5]](#footnote-5)

The reason, and how, each software was used over the development of this project will be explained.

**The Concept**

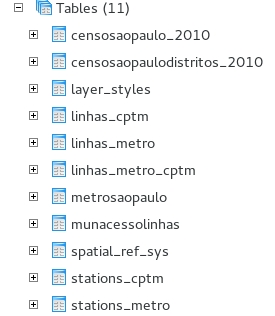
The subject of choice for this web-map was São Paulo’s subway and light rail network. The idea was to convert a static map, which was created by me for a past project, in an interactive web-map.



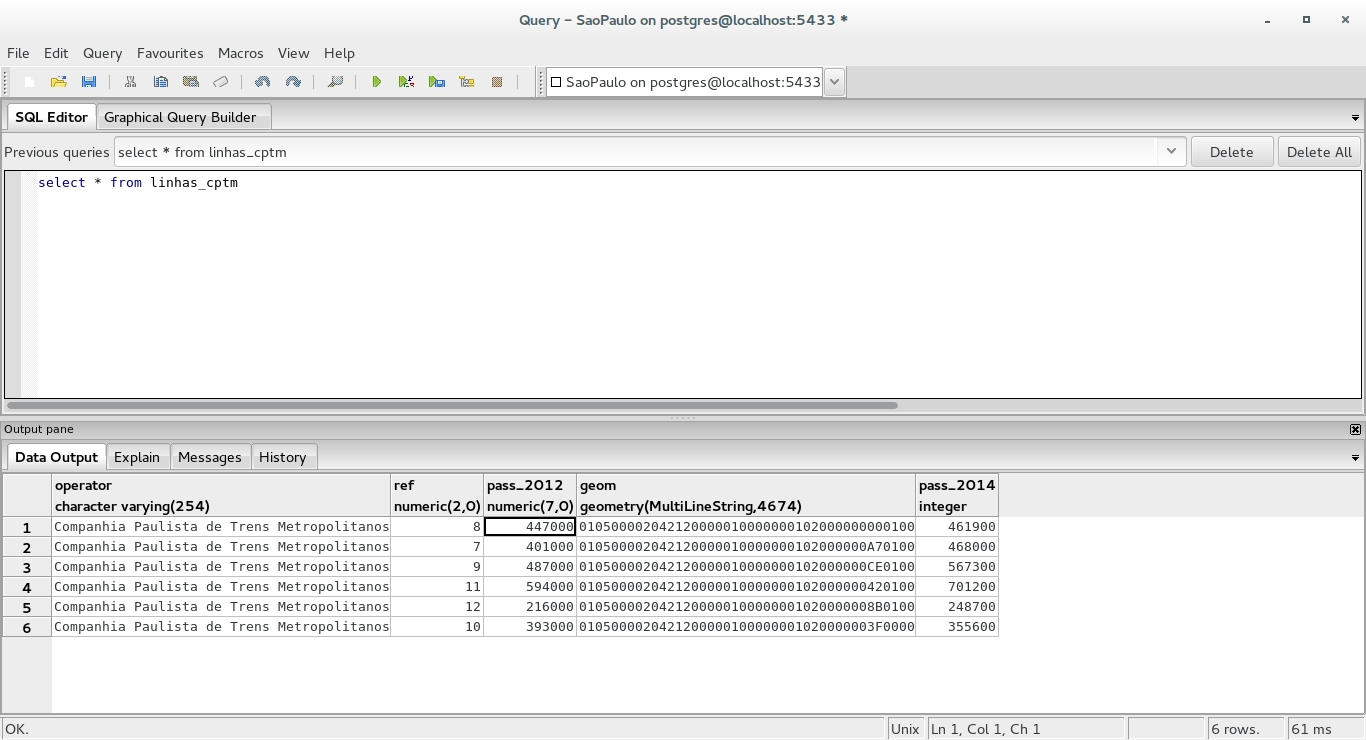
This map shows ridership for all the lines and each color reflects their official color scheme. The final web-map should maintain the same intent.

**The Data**

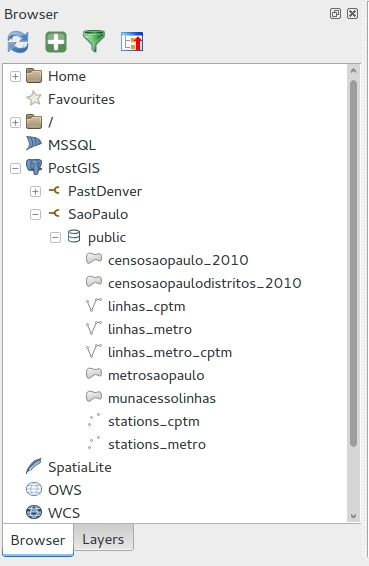
The data was originally gathered from Open Street Maps and cleaned in such way that only one polyline would represent a line. Naturally the OSM data did not have any attribute, such as ridership numbers, for this I compiled numbers from official annual reports and added it to each line. This first step was made in spring of 2014, therefore the initial ridership numbers were from 2012. Once the data was ready to be stored for future use I decided that the safest container would be a database. This is where PostgreSQL and PostGIS came into play. A database was created to hold relevant data about the city of São Paulo. Below are the schemas used on this database:

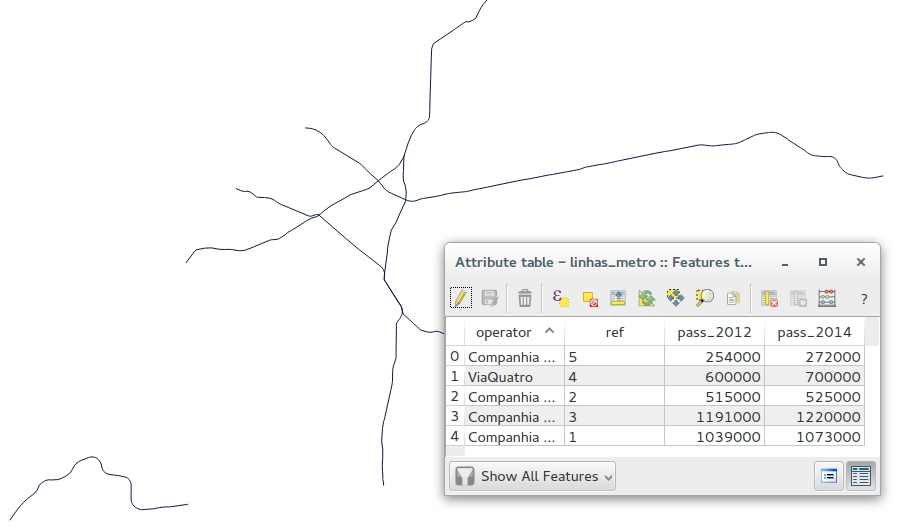


For this project we used the “linhas\_cptm” and “linhas\_metro”. The contents of “linhas\_metro” are the following:



Note that the “pass\_2014” field is the last one, this is because this field was created during the web-map project using the latest official reports. Although I could have add this field using pgAdmin, I decided to edit this data from inside QGIS. QGIS allows users to connect PostgreSQL databases and load data directly into map projects with little overhead. Below is a snippet of how the connection looks like inside QGIS.

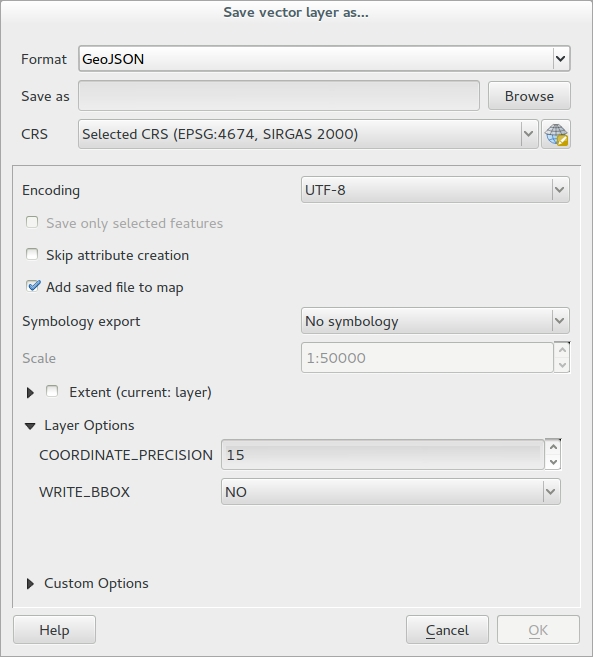




The reason of why I decided to use QGIS to add the new bits of data into the lines geometries was that in QGIS I would be able to visually see if any geometry would need to be edited as well. Luckily the geometries were correct, but QGIS showed another very useful trick for this project. QGIS allows users to export data in GeoJSON format, the reason of why this is important will be explained next.

**The Application Development**

As mentioned before the final product consists of a web-map, for this we choose Leaflet. This JavaScript library offers all the tools to build lightweight “slippy” web maps. Perhaps one of the main advantages of Leaflet is it support to the GeoJSON file format. GeoJSON add spatial capabilities to the JavaScript Object Notation, allowing us to create entire spatial datasets using JavaScript. QGIS offers an option to save a layers as a GeoJSON format right out of the box. All the web-development and design process was conducted in Brackets, which is a very powerful free and open-source software for web-development.



Here is a snippet of how a GeoJSON looks like:

var line\_cptm = {

"type": "FeatureCollection",

"crs": {

"type": "name",

"properties": {

"name": "urn:ogc:def:crs:EPSG::4674"

}

},

"features": [

{

"type": "Feature",

"properties": {

"operator": "Companhia Paulista de Trens Metropolitanos",

"ref": "8",

"pass\_2012": "447,000",

"pass\_2014": "461,900"

},

"geometry": {

"type": "MultiLineString",

"coordinates": [[[-46.640234499999906, -23.533606099999929], [-46.640786399999911, -23.532838899999945], [-46.641319399999873, -23.5320982], [-46.641437499999881, -23.531837399999858], [-46.641526599999963, -23.531597899999952], [-46.6415935, -23.531387799999891], [-46.641682299999957, -23.531177599999978], [-46.641766699999835, -23.531013099999882], [-46.641882399999815, -23.530875799999876], [-46.642327, -23.530254800000023], [-46.642445799999905, -23.530107799999939], [-46.642610099999899, -23.529912399999944], [-46.642762199999879, -23.5297574], **…** [-46.760126299999911, -23.529532599999868], [-46.760186494348602, -23.529583528110948]]]

}

},

The application get its data from two specific GeoJSON files: line\_cptm and line\_metro. Using Leaflet’s L.geoJson function we can easily load these files into our map. The final function looks like this:

//geoJson calls

var cptm = L.geoJson(line\_cptm, {

style: Style,

onEachFeature: onEachFeature

}).addTo(map);

var metro = L.geoJson(line\_metro, {

style: Style,

onEachFeature: onEachFeature

}).addTo(map);

The trick was to style each line using their respective official color. For this I create a function that would go over each GeoJSON and would match a specific color value to a specific number. First the function that would attribute a line number to a color value:

function getColor(d) {

return d == 1 ? "#171796" :

d == 2 ? "#007A5E" :

d == 3 ? "#ED2E38" :

d == 4 ? "#FFD525" :

d == 5 ? "#BA1FB5" :

d == 7 ? "#a50050" :

d == 8 ? "#919388" :

d == 9 ? "#00ab84" :

d == 10 ? "#00778b" :

d == 11 ? "#e05a5c" :

d == 12 ? "#001a72" :

"#b10026";

}

Each HEX color value is based on official color values set by CPTM and Metro. Next I create a function with that would look inside the GeoJSON file take the line number and then match the number to the value on the getColor function:

function Style(feature) {

return {

color: getColor(feature.properties.ref),

weight: 5,

opacity: 1,

dashArray: ‘ ‘,

};

}

This function also set the desired width (weight), and color opacity. The final touches was to add interactivity when hovering over the lines and display the ridership numbers on the left hand side part of the webpage. This section of the map was created by using Bootstrap[[6]](#footnote-6), which is also an open-source project, as a HTML/CSS framework to set the desired area for the map and information.

**Conclusion**

This project shows clearly how different free and open-source platforms can be easily combined to create an interactive web-map application that allow users to explore the information behind the data. Each piece of software on this project had crucial participation, PostgreSQL/PostGIS help us to safely store our data so we can easily retrieve it in the future, QGIS help us to visualize, edit and export our data, and finally Leaflet offers all required tools to build high quality web maps that looks and feel modern.

1. <http://www.qgis.org/en/site/> [↑](#footnote-ref-1)
2. <http://www.postgresql.org/> [↑](#footnote-ref-2)
3. <http://postgis.net/> [↑](#footnote-ref-3)
4. <http://leafletjs.com/> [↑](#footnote-ref-4)
5. <http://brackets.io/> [↑](#footnote-ref-5)
6. <http://getbootstrap.com/> [↑](#footnote-ref-6)