

Web Mapping with QGIS2WEB and Leaflet

The goal for this new lab material is to visualize the final product(s) from your Resilience Academy analytical lesson in the form of a Leaflet map on GitHub. If any of the instructions contained here are ambiguous or could use an accompanying video tutorial, please let me know by Friday at 2:30 pm.

New resource: PostGIS Cheat Sheet: https://www.postgis.us/downloads/postgis20_cheatsheet.html

Creating interactive "slippy" maps (those you can pan and zoom easily) for the internet is getting easier and easier! Two powerful open-source JavaScript libraries for map construction have been game-changers:

- 1) OpenLayers: <https://openlayers.org/>
- 2) Leaflet: <https://leafletjs.com/>
 - a) Documentation: <https://leafletjs.com/reference-1.5.0.html>
 - b) Tutorial examples (a great way to learn Leaflet from the bottom up): <https://leafletjs.com/examples.html>

QGIS has a QGIS2WEB plugin for automatically generating webpage files for Leaflet. I'm most familiar with Leaflet, so let's go with that!

- 1) QGIS2WEB: <https://github.com/tomchadwin/qgis2web>

Web Mapping Technologies

- 1) Vector data is usually stored in GeoJSON format: <https://geojson.org/>
 - a) This is a geographic version of JSON: JavaScript Object Notation
- 2) HTML (hypertext markup language) is the language used to format webpages, based on tags contained in brackets.
- 3) CSS (cascading style sheets) is a notation for styling webpages: think of it as a code version of a set of styles or a template in a program like Microsoft Word.
- 4) JavaScript is a light programming language in which code can be written for interactive web applications.

Using QGIS2WEB

- 1) Before exporting a map to the internet:
 - a) Export or query feature layers with *minimum* of attributes: don't include anything you don't want public or used for styling or popups!
 - b) Symbolize points as simple marker circles for ease of translating to Leaflet `circleMarker` symbols.
 - c) Do not attempt to display all buildings or building centroids: the datasets are too large. To successfully create an interactive map of such a large dataset, you would need Leaflet to be able to connect directly to your database in order to optimize querying only the features in the

current map view, and you would need to make the layers invisible until the map is zoomed closely enough to query only a smaller set of features. This is not possible with our current database setup.

d) Include at least one base layer, e.g. by using **QuickMapServices** to add the **OpenStreetMap** layer.

2) Load the QGIS2WEB plugin

a) Plugins → Manage and Install Plugins →

b) Web → QGIS2WEB → Create Web Map

3) Set the exporter to use **Leaflet** with the option at the bottom of the plugin:



4) **Layers and Groups** tab settings:

a) Check only the layers you want to include in the web map.

b) Your map readers will be able to turn layers on or off. Uncheck the **Visible** option if you want to load a layer into your map, but to start with the layer being invisible until the map reader selects it in the legend.

c) If any of the layers are WFS (web feature service) layers, check the option to create a GeoJSON for them. This essentially makes your own local copy of that data.

d) If any of the layers are WMS or XYZ Tile layers, the exporter will direct the Leaflet map to acquire data directly from the same internet-based WMS or XYZ servers without saving any local data.

e) If any of the layers are local vector layers (e.g. from your own database), then the exporter will convert them to `.js` files using the GeoJSON format.

5) **Appearance tab:**

a) Change **Add layers list** to *expanded* to always list each layer and allow them to be turned on and off

b) Change Template to *full-screen* to create maps that fill the web browser.

c) Scale/Zoom can be customized:

i) **Extent** is setting the default zoom level and extent of the map when it loads in a webpage. Zoom and pan your current QGIS view to where you'd like the map to start, and set this to *Canvas extent*.

ii) **Max zoom level** limits how far the user can zoom in. Most web map base layers do not zoom beyond level 19

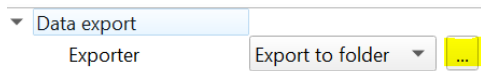
iii) **Min zoom level** limits how far the user can zoom out. The world looks dumb zoomed out too far in world Mercator, and this is a localized project. Zoom level 6 is already large scale enough to allow someone to see all of Tanzania on their screen.

iv) **Restrict to extent** : you may use this to prevent the map from ever panning out of the bounds of the extent currently displayed in QGIS.

d) By default, let's not complicate the map with options for address search, geolocate user, highlight on hover, layer search, project CRS, measure tool, or popups.

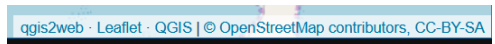
6) Export tab:

- a) Set the **exporter** to *export to folder*, and use the ellipsis button set the folder to a new folder on your `W: \ drive`



- b) Leave other options as default: no labels
 - c) With this setting, the plugin will create a time-stamped folder inside your export folder, containing all the files for your map.
 - d) You can set **precision** to your max zoom level. This can simplify vector geometries to the precision needed to draw them correctly at your largest visualization scale.
- 7) **Update preview** will show a preview of the map on the right site. It can take quite some time to load with complicated vector datasets.
- 8) **Export** to save the map in a time-stamped folder inside the export folder you set above.
- a) It may take some time to preview or export layers with large numbers of features to GeoJSON.
- 9) The export probably opens a webpage of your map.
- a) If not, open the folder you set to the export folder and open `index.html`.
 - b) Enjoy your map for a minute!
 - c) Let's edit final touches of the map before uploading to GitHub.

Editing the HTML page

- 1) Open the `index.html` file for your map in Notepad ++
 - a) Each time you make a change to this file, I suggest saving it and refreshing the page in your web browser, to see if the change worked.
- 2) Search for the `map.attributioncontrol.SetPrefix` function.
 - a) This function creates the text shown at the bottom-right of your map:

 - b) Anything in the prefix will *always appear*, and additional attributions for individual layers will appear only when the layer is activated. Try turning layers on and off for your map: if you have an OpenStreetMap layer, its attribution probably appears and disappears with the layer.
 - c) All maps should have a prefix crediting qgis2web, Leaflet, and QGIS as technologies used to create the map.
 - d) Notice the whole prefix is enclosed in single quotes.
 - e) Notice each attribution is actually HTML code for a link so that you can click the attribution and go to its webpage. For example: `QGIS`
 - i) `` is the beginning of the link, where `href=` defines a web address, in double quotes
 - ii) `QGIS` is the text shown on the map for the link
 - iii) `` ends the link.

- iv) Notice how other links include `target="_blank"` ? Try clicking the QGIS link, and then clicking the Leaflet link. Notice that one opens a new window/tab and the other does not?
- v) Try adding `target="_blank"` to the QGIS link:

```
<a href=https://qgis.org target="_blank" >
```
- vi) Save the HTML file and refresh your map and try to see if this forced the QGIS link to also open a new tab.
- vii) Note: `·` is code for the small dot between attributions, but most are separated by `|` (the vertical bar on your slash \ key)

- f) Try changing the prefix to start with your name and a link to your GitHub account. For example, I have revised my prefix function to:

```
map.attributionControl.setPrefix(
'

```

- i) I did not add `target="_blank"` to my own link, because I want users to be able to go from my GitHub page to my map and back again without opening a new tab.

- 3) Search for `attribution` and be sure to give appropriate credit for all layers.

- a) Tile layers like OpenStreetMap have this filled automatically, as an attribute in their layer definition. E.g. my OpenStreetMap layer has a line:

```
attribution: '

```

- b) I will change the line for my Subwards layer from `attribution: ''`, to

```
attribution: '

```

Optional Leaflet Customizations

- 1) Would you like a scale bar on the map?

- a) At the end of the document, *just before the final* `</script>` tag to end the JavaScript, add a line:

```
L.control.scale().addTo(map);
```

- b) Scale can be further customized, referring to <https://leafletjs.com/reference-1.5.0.html#control-scale>

- c) For example, optionally a set of parameters with one to remove imperial measurements and keep metric measurements:

```
L.control.scale( {
  imperial: false,
  metric: true
} ).addTo(map);
```

- 2) Each vector layer has a function to define its style, and if the layer is classified, it will have a series of *if* conditions to decide first which class the feature is in, and next which style to apply. You may

experiment with the following variables, some of which apply only to circle markers and some of which only apply to lines or polygons. Find these functions by searching for `function style_`

- a) *Radius*: size of circle markers.
 - b) *Opacity*: with 1 being opaque and 0 being transparent.
 - c) *Color*: is the color of a line or outline, with an `rgba()` function
 - i) First number is Red, on scale from 0 (no color) to 255 (full color)
 - ii) Second number is Green, on scale from 0 (no color) to 255 (full color)
 - iii) Third number is Blue, on scale from 0 (no color) to 255 (full color)
 - iv) Fourth number is transparency, on scale from 0 (completely transparent) to 1 (completely opaque)
 - v) For help on choosing colors, use https://www.w3schools.com/colors/colors_picker.asp to find the red, green, and blue values you want.
 - d) *Weight*: is the width of a line or outline
 - e) *fillOpacity*: is the opacity of a marker or polygon fill
 - f) *fillColor*: is the color of a marker or polygon fill
 - g) *interactive*: values of `true` and `false` control whether pop-ups will be functional for each feature and each layer. True is functional, false is disabled.
- 3) If you have enabled feature pop-up content, you can change its formatting. Search for `popupContent`.
- a) The popup content creates a text string of HTML code enclosed by single quotes.
 - b) It sets up a table with a row for each attribute. Do you see the begin table `<table>` tag and end table `</table>` tag?
 - c) Each row contains one attribute and begins with `<tr>\` and ends with `</tr>\`
 - d) You can delete any rows you don't want to see in the popup, or if you know HTML, format the popup any way you wish.
- 4) You can customize the initial map extent, maximum and minimum zoom levels, and limits to how far the user can pan the map.
- a) Near the top of the HTML document, the map itself is initialized with a new variable:
`var map = L.map('map', { ... })`
 - b) Within the `{}` object, you can set the `maxZoom` and `minZoom` levels, as long as `zoomControl:true` exists to turn on zooming control.
 - c) The `L.map()` function is followed by a `.fitBounds` function, giving the lower-left and upper-right coordinates of the initial map view in decimal degrees.
 - d) For example, in the map initialization below, the map must not zoom in beyond level 19 or out beyond level 6, and initializes by viewing an area bounded by 7.154 degrees South, 38.786 degrees East, 6.508 degrees South and 39.851 degrees East:

```
var map = L.map('map', {
    zoomControl:true, maxZoom:19, minZoom:6
}).fitBounds([[[-7.154, 38.787], [-6.508, 39.851]]]);
```

e) If you want the user to be limited to panning beyond a set limit, find the function `setBounds()` { ... } and change its contents to:

```
function setBounds() {  
    map.setMaxBounds([[ -7.154, 38.787], [ -6.508, 39.851 ]]);  
}
```

You may change the bounding coordinates to any region you need, and I suggest starting by copying the coordinates from the map extent in the `fitBounds()` function above.

Uploading to GitHub

- 1) Rename the folder containing your Leaflet map to something simple with no spaces, e.g. `dsmap` (for Dar es Salaam map)
 - a) Note: it is nearly impossible to delete an entire folder with contents using GitHub online. Therefore, I really suggest using a simple consistent name for your Leaflet map(s) and if you need to modify that, you can upload again and overwrite the previous map.

- 2) Log into your GitHub account and navigate to your `.github.io` repository

- 3) Go to the **Upload files** button



- a) Drag the *folder* onto gitub's file upload window.
 - b) Once all the files have uploaded, **commit changes**
- 4) Edit your `index.md` file to add a link to your map. For example, mine is:
`[Leaflet map of pharmacies in Dar es Salaam](dsmap/index.html)`
 - 5) Congratulations! You've made an interactive web map with the Leaflet JavaScript library!

Learning from others' Leaflet maps

- 1) If you want to learn more, start by looking at the Leaflet tutorial maps:
<https://leafletjs.com/examples.html>
- 2) For any leaflet map you see on the internet, right-click the page and **View Page Source** to see their code. For example, the first simple leaflet tutorial <https://leafletjs.com/examples/quick-start/> has links to "see this example stand-alone" below each sample. Open those links and view the page source to see how the example was made.
- 3) This instruction is for Firefox, but all web browsers tend to have a similar option.

