Lab 03: Your First Web Map

> In this lab you will walk through demonstration steps to create an interactive web map. You will be asked to create your own custom web map using these steps. You may wish to scroll to the end section and review the deliverables for this lab before getting started.

Objectives

- Get comfortable setting up a development environment
- Introduce a simple working web map template
- Complete a mapping scenario of a route from one location to another
- Refining the design

Let's go 👇

Part 1: Tutorial

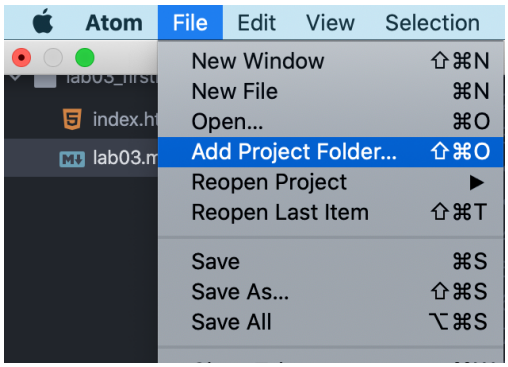
Setting up a development environment

The first hurdle in webmapping is getting your development environment set up to write code, develop, and test the map. To get started, open up `Atom` and then setup a working directory somewhere on your computer. You must use the following organization within this directory.

```
[your_repository_name]
    |index.html
    |readme.md
    |--data
    |       [name of your data file].geojson
    |       [name of your data file].geojson
```

- The repository or directory name is a folder called "lab03_firstmap" or similar.
- `index.html` is a HTML file that contains a webmap template for use in this lab. You can download it from the course webpage Lab 3 assignment.
- You will create a `readme.md` file in Atom once you are finished with your webmap that describes your application using Markdown language.
- A `data` folder will contain any separate geographic data files for use in our webmap.

In Atom, when beginning a new project, make sure that you add the project folder of the directory you first created, so you can view these files and folders as you work in your development environment. This step is important so that Atom can recognize folder and file locations.

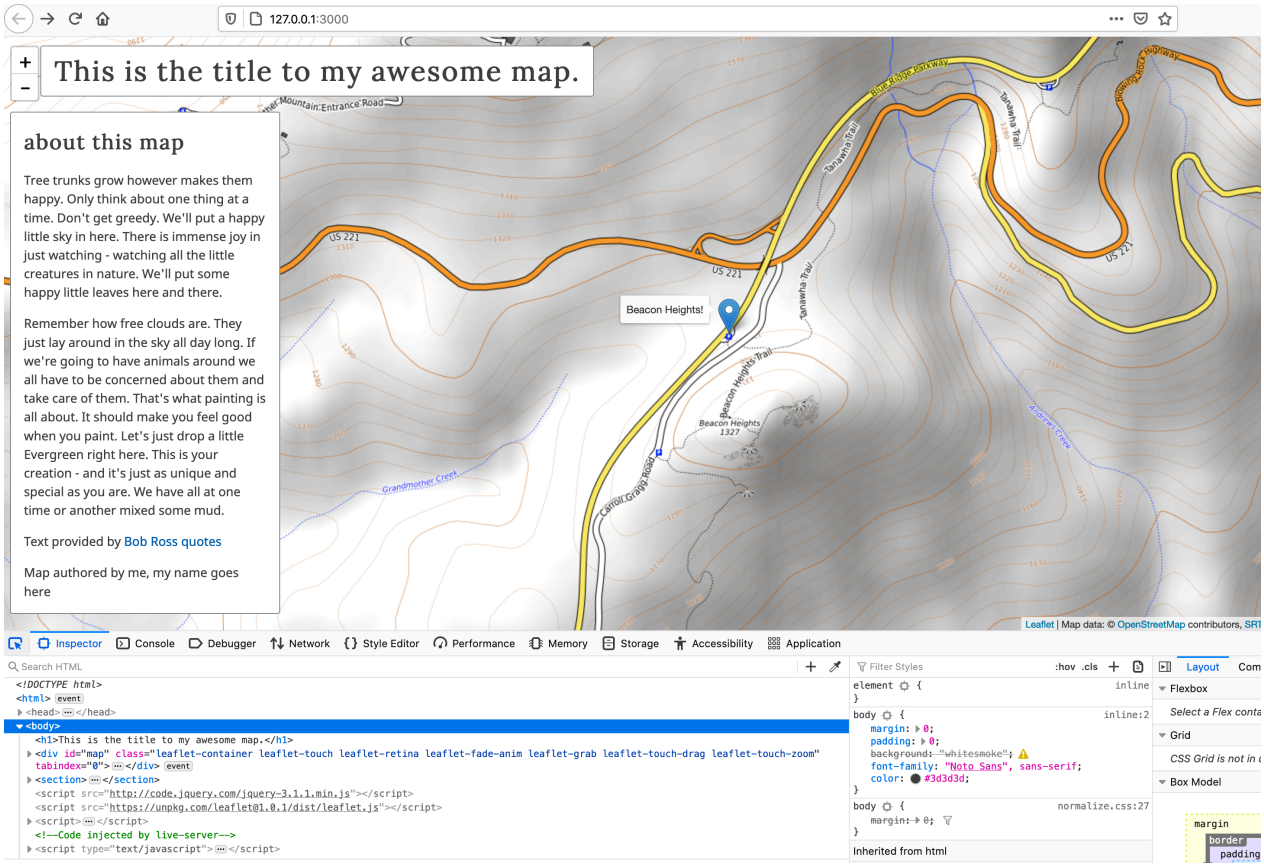


> Adding a project folder in Atom.

Now, open `index.html` in Atom by double clicking it in your project pane. We will also launch Atom live server by using `Ctrl` + `Alt` + `l` (`Ctrl` + `Option` + `l` on Mac) in Atom's main window once the HTML document is opened. The url address of index.html should be `https://localhost:3000/index.html` , or something similar based on how you have structured your directory.

As a rule of thumb, you need to remember when the server is live and when it is not. When you make and save changes to your HTML document, the browser should automatically apply those changes over the live server, which will show you the result. To close the live server, remember that you can use `Ctrl` + `Alt` + `q` (`Ctrl` + `Option` + `q` on Mac).

Open the browser web developer tools as you did in a previous exercise and use the Elements tab to inspect the DOM as it is rendered within the browser. This section will largely mirror the HTML document and you should also see elements that are dynamically produced with JavaScript when the page loads.



> Exploring the web developer tools highlights elements of the HTML Document.

Exploring the web map template

If the map loads correctly, you should see a topographic basemap centered on Beacon Heights on the Blue Ridge Parkway, just outside of Boone. There is also a marker at this location and a tooltip opens on the marker. There should be no errors within the Developer Tool Console.

How does the code contained within `index.html` work to produce this map? The HTML document is composed of 3 essential web technologies: HTML, CSS, and JavaScript, so the web browser creates a DOM or Document Object Model using this document as a guide.

HTML (structure)

HTML structures and describes the content of our document, or webpage. Some of the elements (`<style>` and `<script>` tags) load additional files such as CSS, JavaScript, or images into our webpage upon page load.

For an example, study this skeleton HTML template:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset=utf-8 />
    <title>Leaflet Map Template</title>
    <meta name='viewport' content='initial-scale=1,maximum-scale=1,user-scalable=no' />

    <!-- load additional CSS files here and this is an HTML comment -->

    <style>

        /* CSS rules go here and this is a CSS comment */

    </style>
</head>

<body>

    <h1>This is the title to my awesome map.</h1>

    <!-- more HTML goes here -->

    <!-- load additional JS files here -->
    <script>

        // JavaScript goes here and this is a JS comment

    </script>

</body>

</html>
```

> Read more about HTML.

CSS (style)

If the HTML structures the content, the role of CSS is to give style to that structure. We apply CSS rules to the webpage and elements we draw on the map to adjust its look and feel.

Study the section of CSS below, which belongs within the `<style>` tags of the HTML document above.

```
body {
    margin: 0;
    padding: 0;
    background: "whitesmoke";
    font-family: "Noto Sans", sans-serif;
    color: #3d3d3d;
}

h1 {
    position: absolute;
    margin-top: 0;
    top: 10px;
    left: 45px;
    font-size: 2em;
    font-family: "Lora", serif;
    letter-spacing: .04em;
    padding: 10px 15px;
    background: rgba(256, 256, 256, .4);
    border: 1px solid grey;
    border-radius: 3px;
    z-index: 800;
}

#map {
    position: absolute;
```

```
    top: 0;
    bottom: 0;
    width: 100%;
}
```

> Read more about CSS.

JavaScript (behavior)

JavaScript is used to add event-driven behavior to our webpage. These interaction behaviors make it dynamic and much more exciting than static maps. Study the following JavaScript, which creates our map within the `<script>` tags of our HTML document.

```
//options to be used when creating the map
        var options = {
            center: [36.08403, -81.83015],
            zoom: 16
        }

//creation of the Leaflet map
        var map = L.map('map', options);

//request to load basemap
var OpenTopoMap = L.tileLayer('https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png', {
maxZoom: 17,
attribution: 'Map data: &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributo
}).addTo(map);

//string content to be inserted into a tooltip
        var message = 'Beacon Heights!';

//create a Leaflet marker, centered on the map's center.
        L.marker(map.getCenter())
            .bindTooltip(message) //bind the tooltip and message to the marker
            .addTo(map) // add the marker to the map`
            .openTooltip(); // open the tooltip
```

> Read more about JavaScript.

Review what the JavaScript above is doing:

1. The statement assigning an object to the variable `options` is specifying Leaflet's map object's options, centering the map on Beacon Heights and setting the Zoom level to 16.
2. The next statement actually creates the Leaflet map object, places it into a DOM element with the `id` value of `map` (for example, `<div id='map'></div>`) within the HTML and applies the options specified.
3. The statement assigning a value to the variable `tiles` allows us to choose a basemap from a remote server. There are many choices you can use at Leaflet Providers. You can see we have chosen the OpenTopoMap example. Explore some of the options to see how the JavaScript for a custom basemap is copied and pasted into your script. You may want a specific basemap for your custom map. Some of them are password protected with a key and will not be publicly available.
4. The last two statements assign a string value `Beacon Heights!` to the variable `message` and uses the Leaflet L.marker class to place a marker at the center of the map, as well as the Leaflet L.Tooltip class to display the message on top of the map layers.

Overall, the Leaflet API Reference is the key to understand how Leaflet operates under the hood.

Now we will turn back to our template `index.html` document to create a basic thematic map.

Mapping a hiking route from Beacon Heights to Price Park on the Tanawha Trail.

Let's begin customizing our web map by drawing a route from Beacon Heights to Price Park using the Tanawha Trail, a popular High Country Destination. ⛰️ The goal of the map is to allow the reader to see the starting and ending points, the trail route

between them, as well as a couple of overlooks and trailheads along the way. For example, a reader might want to stop at Rough Ridge or the Daniel Boone Scout Trail before continuing on to Price Park.

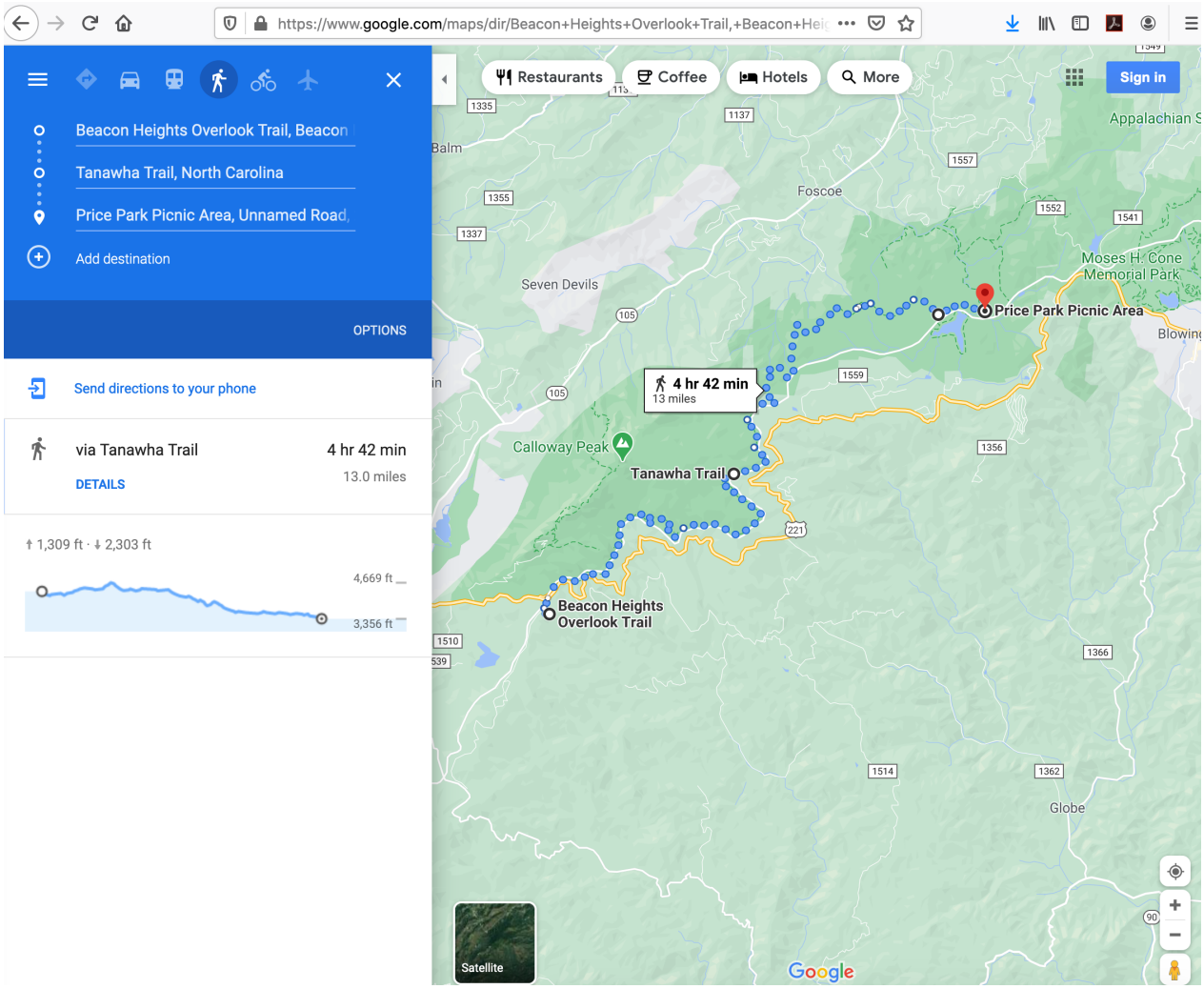
Now that we've conceived our geographic purpose or design for the map, we need to capture some appropriate data and then display it in a web and mobile friendly format. Additionally, we want the reader to retrieve specific information about these places by interacting with the map.

Step 1: Data acquisition and conversion

To get data and convert it to an appropriate format for web mapping, we will use a few web-based tools.

One of the best tools available is Google Maps!

1. First, do a simple Google Map search for the route end point, Price Park Picnic Area.
2. Use the Directions functionality to determine the route from Beacon Heights to Price Park. Select the Pedestrian mode of travel and add an intermediate destination of the Tanawha Trail.
3. To shortcut these steps, the map is shared here.

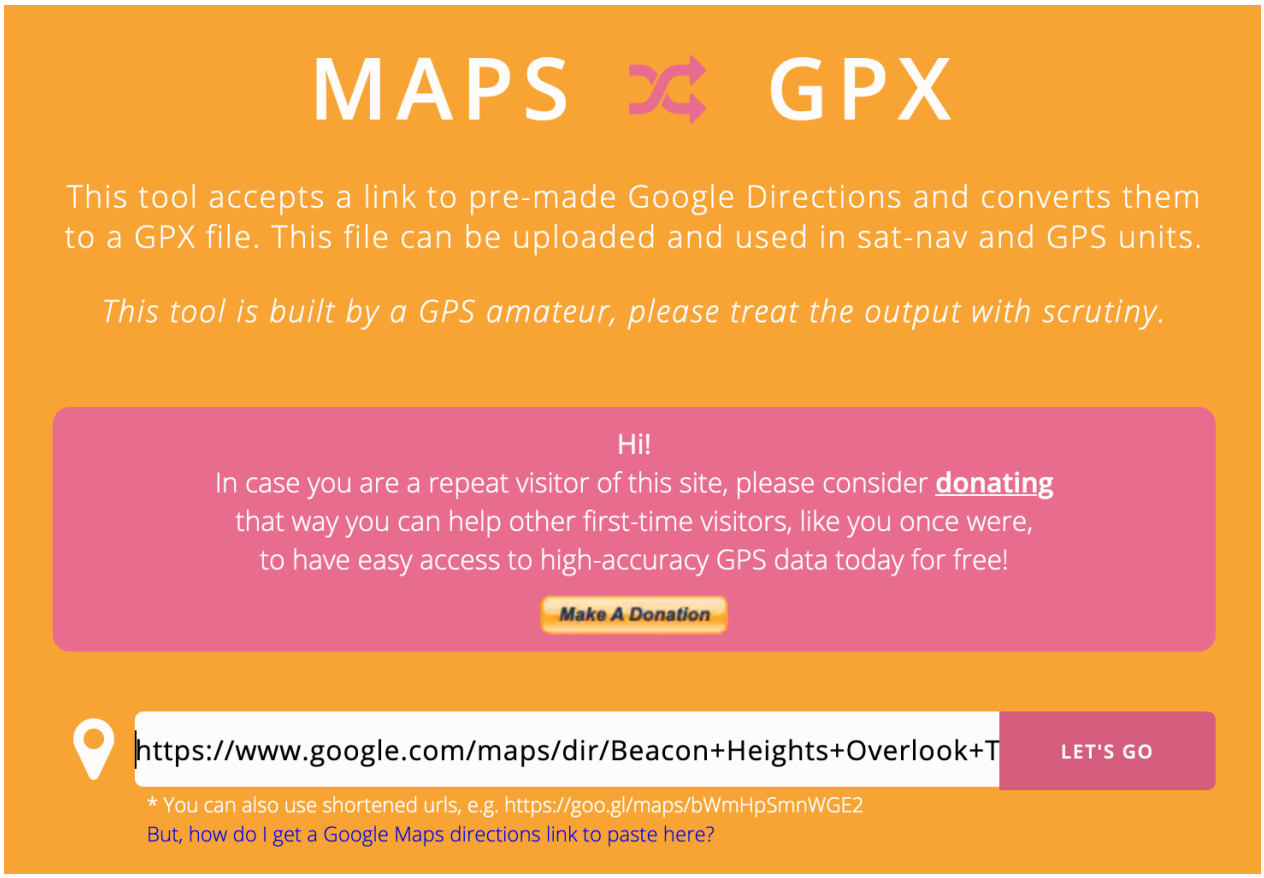


> Using Google Maps to find the route from Beacon Heights to Price Park via the Tanawha Trail.

Google Maps allows you to pick between alternative routes, as well as to drag the route to customize the desired path.

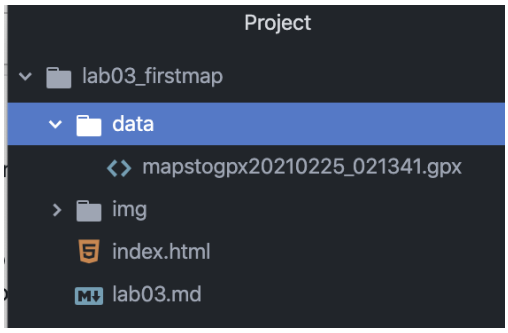
Once you have highlighted the desired route in blue on the map, copy the entire URL from the address bar (highlight it, and select `Edit -> Copy` or use `Cntr +C`).

4. Go to a website called Maps to GPX, which provides a tool that "accepts links to pre-made Google Directions and converts them to a GPX file." Paste your URL from Google Maps into the form and hit "Let's Go."



> Using Maps to GPX to convert a Google Maps route.

The site will do the file conversion and then prompt the GPX file to download. Now, move this file into the `data` folder that you created in your project directory at the beginning of the lab.



> Sample location of this file in your project directory in Atom.

GPX (GPS Exchange Format) is a text-based format derived from XML and often used to encode GPS data. You can open this file in your text editor to examine the contents.

If you happen to be a Strava user, you can also download all your routes in GPX format.

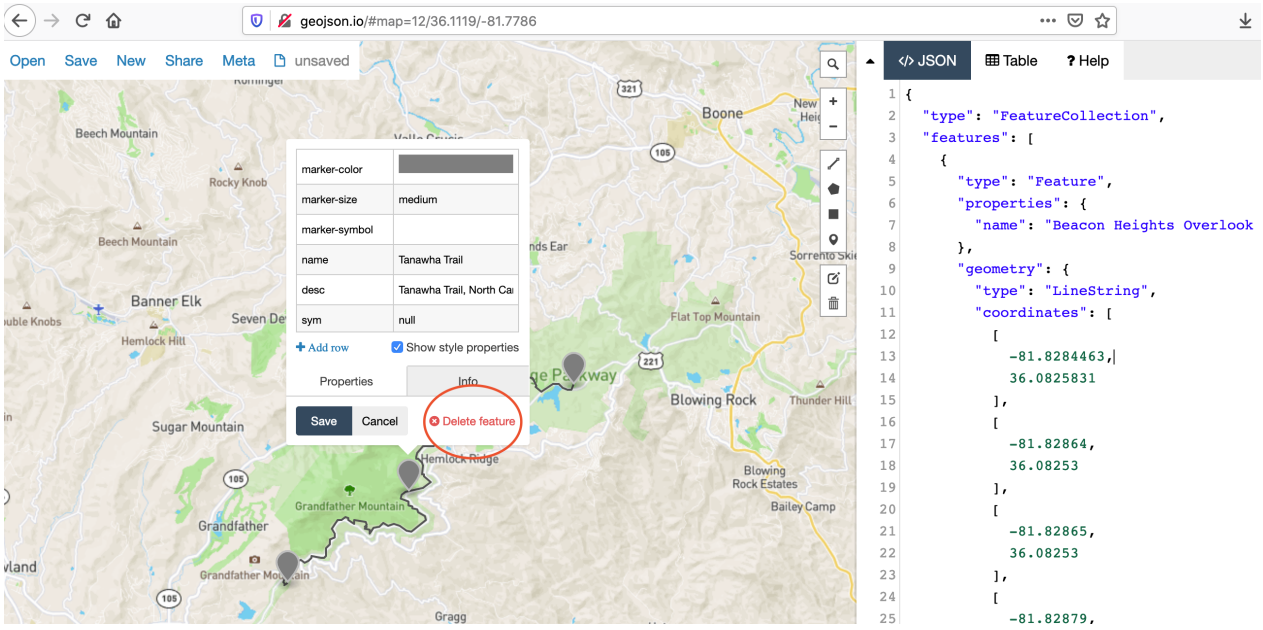
5. Next, let's convert the data to another format: GeoJSON. The GeoJSON is to webmapping what the shapefile is to GIS.

   Navigate your browser to a website called geojson.io. We will use this alot, so bookmark this page.

Open your GPX file in the geojson.io web application. Study the code generated in the right panel. It is the GeoJSON encoding of your route. Unlike shapefiles, GeoJSON can encode multiple geometry types within a single Feature Collection. Note that Features have both `properties` and `attributes`. The `"Linestring"` type contains all the points that make up the route, while any `"Point"` type Features encode the endpoints of the route.

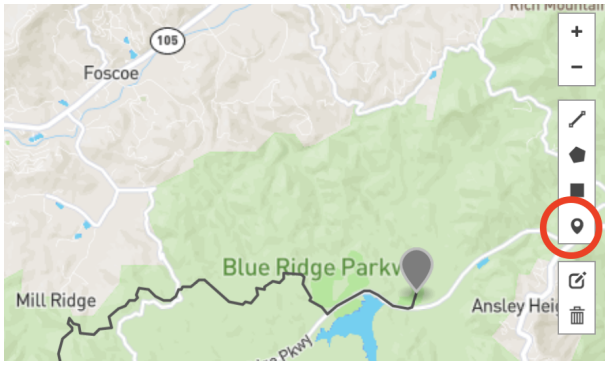
Explore your data with the geojson.io website.

Note that our route retained some attributes from Google Maps that we don't need. We can remove these attributes manually and edit any data properties. We could also add, remove, or edit geometries to modify your route.

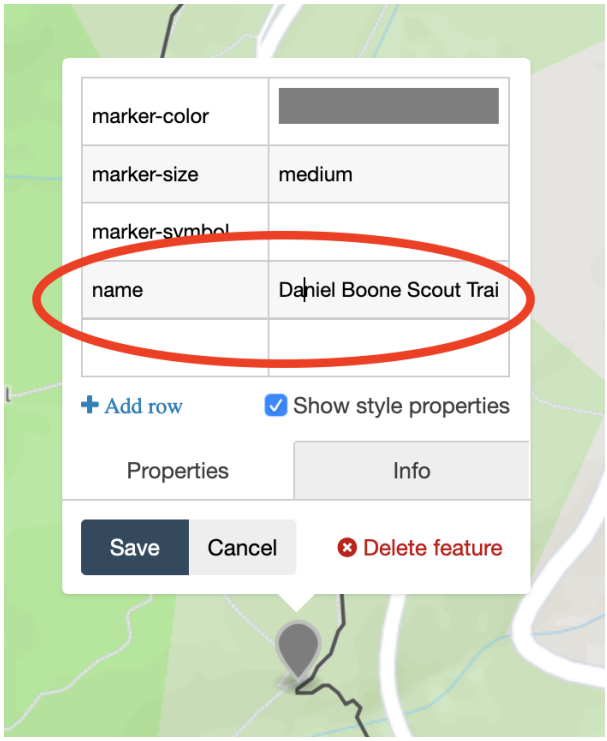


Removing the Tanawha Trail marker in geojson.io.

Instead, let's add a couple more places of interest. Using the drawing tools, place a point of interest along your route. Add a property row to the marker, and be sure to use the word "name" as the name of the attribute (just like the other points).

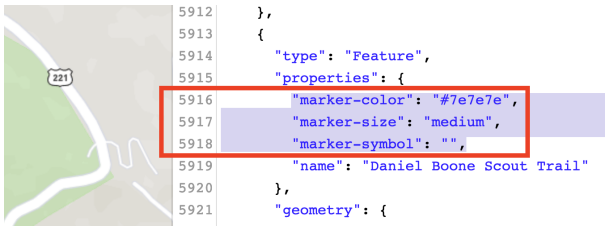


Draw a new Marker icon.



> Add property row, with "name" and name of marker.

Geojson.io also adds some other properties to style the marker, as you can see above "marker-color", "marker-size", "marker-symbol". We don't need these, and you can remove these lines in the editor.



> Lines to delete. Note we're all the way down at line 5916 of our GeoJSON file because there are alot of trail coordinates on the route.

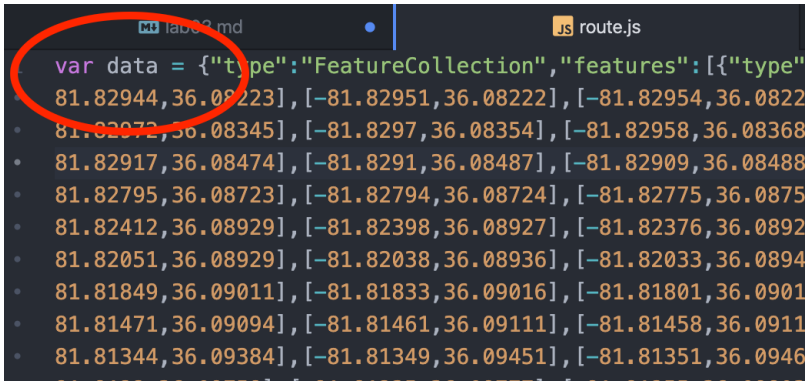
Once you've completed edits to your data, choose **Save** and download the GeoJSON (it will download with the filename `map.geojson`). Then move the file into your `data` folder within your project directory.

You can open your geojson file in a text editor to confirm it looks the same as it did on geojson.io. Next, we will load the data into the web map.

Step 2: Loading external data onto the web.

There are several ways to load GeoJSON data into a web map. While the best (and most sophisticated) way is to make an AJAX request, we will begin with a more simple solution. We will use AJAX later in the class.

First, rename the `map.geojson` file to `route.js`. Then open the `route.js` file in Atom by double clicking it from the project pane and assign the entire GeoJSON structure (all of the code) to a JavaScript variable named `data`:



> Creating a JavaScript variable of your GeoJSON file.

Save `route.js` in Atom with those changes. GeoJSON is basically a JavaScript object, and now we've assigned it to a variable called `data`, which we can now reference within our script.

Move back to `index.html` in Atom. Scroll down to the JavaScript section of our document to where the `<script>` tags are located. There are currently two `<script>` elements, which load remote jQuery and Leaflet JavaScript files before our custom JavaScript code executes. These are required to load Leaflet and jQuery. Now, load our new JavaScript file into our document, being careful to specify the appropriate relative path to our file within our project directory.

Add the line `<script src="data/route.js"></script>` to our index.html file, beneath where the Leaflet and jQuery libraries are loaded but **before** the `<script></script>` tags that enclose our custom JavaScript.

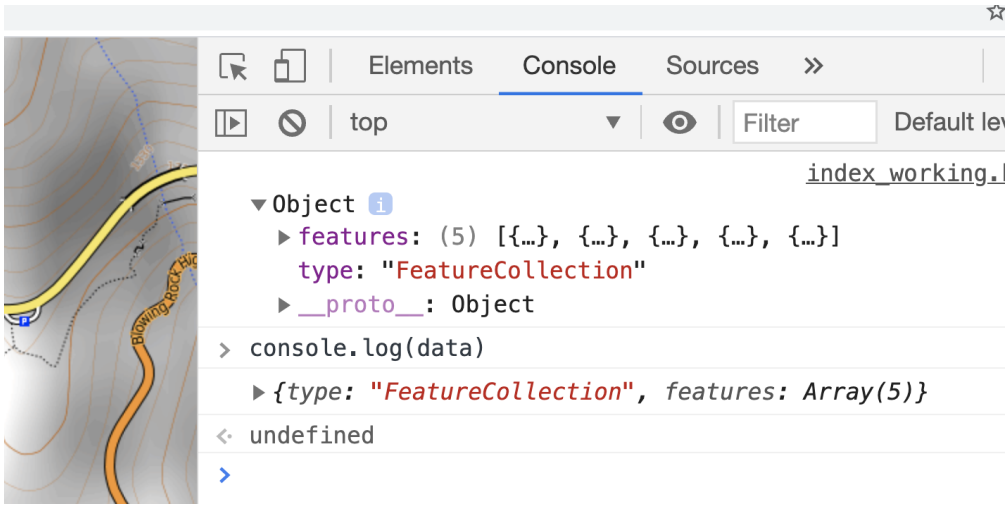
```
98    <script src="http://code.jquery.com/jquery-3.1.1.min.js"></script>
99    <script src="https://unpkg.com/leaflet@1.0.1/dist/leaflet.js"></script>
100
101   <script src="data/route.js"></script>  //OUR DATA LOADED HERE!
102
103   <script>
104
105 //options to be used when creating the map
106     var options = {
107       center: [36.08403, -81.83015],
108       zoom: 16
109     }
```

> Loading our data into the JavaScript section of the HTML document.

You can verify that the data are loaded and you have access to the JavaScript object `data` we created by using a `console.log()` statement and then looking in the Console Web Developer Tools in your browser.

```
103    <script>
104
105  //options to be used when creating the map
106      var options = {
107        center: [36.08403, -81.83015],
108        zoom: 16
109      }
110
111      console.log(data); // output will be our GeoJSON object
112
113  //creation of the Leaflet map
114      var map = L.map('map', options);
```

Adding console.log(data); to our custom JavaScript



Confirmation that our data are loaded.

Now, time to draw the map!

Step 3: Adding GeoJSON to the map.

Data are loaded and Leaflet JavaScript library is available to us in our script, so we're ready to do the actual mapping. Leaflet makes this process very straightforward with its method `L.GeoJSON`.

First, comment out the following code from the template by adding `//` before each statement:

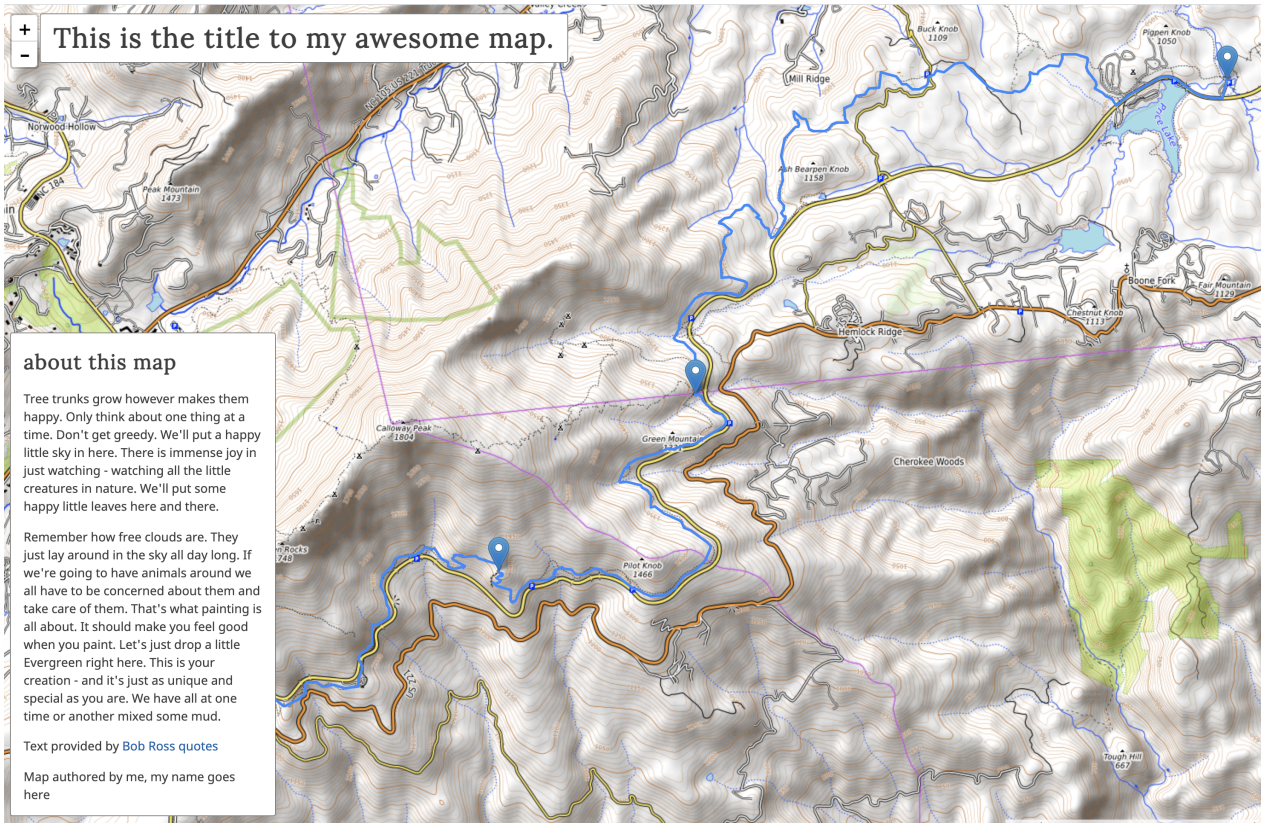
```
//string content to be inserted into a tooltip
//        var message = 'Beacon Heights!';

//create a Leaflet marker, centered on the map's center.
//        L.marker(map.getCenter())
//            .bindTooltip(message) //bind the tooltip and message to the marker
//            .addTo(map) // add the marker to the map`
//            .openTooltip(); // open the tooltip
```

Next, write or paste the following statements beneath that commented out code:

```
var myRoute = L.geoJson(data).addTo(map);
```

```
map.fitBounds(myRoute.getBounds());
```

Save your `index.html`, refresh the browser, and you should see that Leaflet has drawn your data to the map. You can open your developer tools to confirm that there are no errors within the Console. You may need to readjust the pan and zoom levels in the script to see the full extent of your data.



> The GeoJSON data are successfully added using Leaflet's default styling options, which includes a LineString feature and Point features on top of the line using a markerPane.

Step 4: Styling the map.

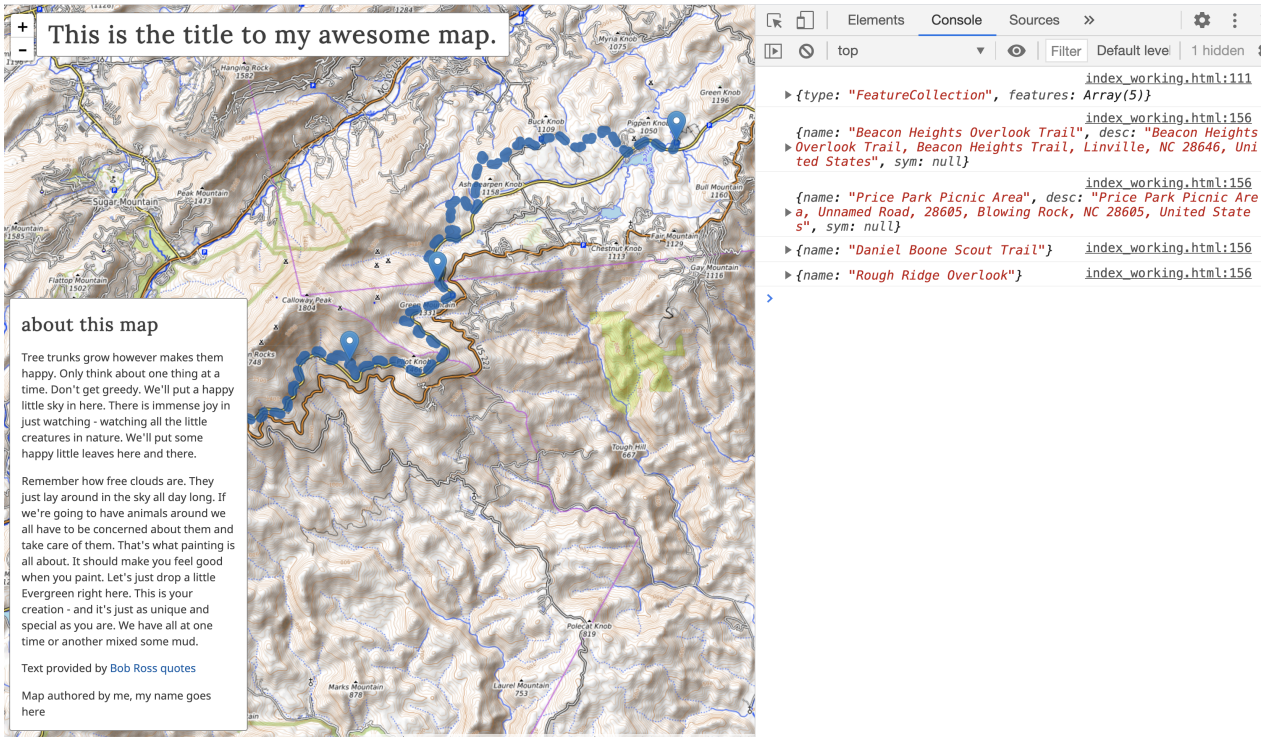
We want to style the line representing our route on the Tanawha Trail and the markers representing our important spots, so we need to separate them into different objects within our script. We can call `L.geoJson()` again to filter the data based on geometry type. To do this, we will use L.GeoJson's filter method.

Replace the line `var myRoute L.geoJson(data).addTo(map);` with the following code snippet:

```
var myRoute = L.geoJson(data, {
  filter : function(feature) {
    if(feature.geometry.type =="LineString") {
      return feature;
    }
  },
  style : function(feature) {

    return {
      color: "#005DAA",
      weight: 6,
      opacity: 0.2,
      dashArray: "5, 5"
    }
  }
  }).addTo(map);

  var myStops = L.geoJson(data, {
```

```
    filter : function(feature) {
      if(feature.geometry.type == "Point") {
        return feature;
      }
    },
    onEachFeature : function(feature, layer) {
      console.log(feature.properties)
    }
    }).addTo(map);
```

Now save your file and refresh your browser.



> The map line feature is now added and styled with Leaflet Path options. In addition, by viewing the console, you can see that our code has logged 5 values, which were accessed through `feature.properties` as L.GeoJson looped through each of the features with `onEachFeature`.

In our styling, we drew the line accroding to the following rules:

```
color: "#005DAA",
weight: 6,
opacity: 0.2,
dashArray: "5, 5"
```

These are called Leaflet styling options, and they are inherited from L.Path class. Play around with these values to change the line symbolization. For example, you could change the color with a different hex code, increase or decrease the line weight, change the opacity, or customize the dashes.

Step 5: Adding user interaction

Replace `console.log()` statement within our JavaScript with the following statement:

```
layer.bindTooltip(feature.properties['name']);
```

Now when you save your document and refresh the browser, you should see that the user is able to retrieve specific information about these features by mousing over a specific feature.

> Inspecting the interaction on each placemarker.

Step 6: Refining the map.

The last step is to make the map complete in order to polish the design and clarify the message and experience for the reader. Adding information about the map through descriptive (or catchy) titles and side columns helps guide your reader's understanding of the map and its purpose. Be sure to update the following.

- Map title
- Map description - get rid of those Bob Ross quotes
- Map author [your name]
- Any useful links to more information about the route
- Adjustments to map zoom levels and centering locations on page loads.
- What else could you change about the design?

Part 2: Lab Deliverables

Description

After successfully deploying this trail map, your task for this lab is to design, develop, and deploy another similar map of your choosing. It must be a route map, which includes a Line Feature and Placemarkers along the way. I encourage you to think creatively. Here are some ideas:

- Your walk to classes with stops at any landmarks, like coffee shops, etc.
- A favorite running, walking, hiking, mountain biking, or skiing route.
- Driving directions for a scenic drive or route to another city.
- Landmarks in a major tourist or travel spot.

Be careful because your assignment will be submitted in a publicly-available GitHub repository, so probably best not to include your home address.

Web map components and rubric

You will be graded on the following components.

1. **(10 points)** - Successful acquisition, conversion, and loading of the GeoJSON of your route with at least 2 landmarks. Pre-obtained GPX files from Strava or similar are acceptable.
2. **(10 points)** - Appropriate basemap loaded from Leaflet Providers. *Tip: Maps of urban areas or roadways are best symbolized with light shaded or dark shaded basemaps, like* `CartoDB.Positron`*, because they reduce visual clutter and make the line features stand out in a complex environment. Trail maps, on the other hand, do well with some sort of topographic or aerial photo basemap.*
3. **(10 points)** - Interactive tooltips are provided on 4 placemarkers for route beginning and endpoints, as well as 2 middle places of interest.
4. **(10 points)** - Customized line symbolization, including color, weight, opacity, and dash options.
5. **(15 points)** - Refined and polished and working map design, including custom title, description, at least one link, map author name, and appropriate zoom and centering levels.
6. **(10 points)** - Write a ~150 word project description as a `readme.md` file using Markdown langugae. The description should include the project name, a brief introduction, major functions, libraries, and data sources of the map. The `readme.md` file should then be saved and uploaded to your project folder, as mentioned at the beginning of the lab.
7. **(25 points)** - Upload your web map project, including all files and folders in the appropriate structure, to a new GitHub repository. Make sure the web map is shareable from a public URL of the GitHub page where the map is published.
8. **(10 points)** - Brief ~150 word reflection of your first web map; what you learned, how it went, what you're looking forward to now.

Some tips for publishing your map through your GitHub account.

You may wish to check out previous lecture or lab documents from when you created a Markdown resume and then published it over GitHub pages.

The internal structure of your project directory in this lab is such that you should be able to upload the entire project folder as a new GitHub repository. First, create a new GitHub repository. Please make sure the name of your repository is **NOT** `lab03` or similiar, use a name which can describe the theme of the map you will make. You want to show your work to a potential employer or graduate school advisor, so it needs to have a polished look and feel in its presentation.

The internal structure of the files and folders in your project repository should be organized and follow the pathnames that you created in your code. The structure below is required and must contain the listed files.

```
[your_repository_name]
    │index.html
    │readme.md
    ├─data
    │      route.js
```

Remember to make your repository public, and also adjust the GitHub Pages settings so that your map is published and viewable over the web.

To Submit on ASULearn

Please submit the following items in the text submission box for Lab 3 on ASULearn.

> - (1) URL of your GitHub Repository
>   (format: https://github.com/[github_username]/[map_repository_name])
> - (2) URL of your published web map on GitHub Pages:
>   (format: https://[github_username].github.io/[map_repository_name])
> - (3) Text reflection of your first web map.