GGR472 Week 5: JavaScript for Web Maps

Class exercise: Creating your first web map with Mapbox GL JS

The Mapbox GL JS is a JavaScript library that allows you to develop web maps and web applications programmatically. Events and methods within the library allow you to display Mapbox maps (including predefined and customised map styles), add data layers, and ultimately incorporate interactivity.

In this exercise, you will first initialize a web map in the browser. Next, you will add data sources to the map and visualize data as layers.

The exercises are ungraded but their primary purpose is to introduce you to basic programming knowledge and skills which may be applied in your labs and final web map project. You may therefore use these instructions and your output code as resources when working on future submissions.

Step 1: Set up HTML file

In a new working directory in Visual Studio Code, create a basic HTML web page (*index.html*) using the boiler plate HTML code. Set the title of the web page to "My first web map".

Using the Mapbox CDN (content delivery network), add the Mapbox library and CSS files into the <head> tag of your HTML file.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My first web map</title>
    <script src='https://api.mapbox.com/mapbox-gl-js/v3.1.2/mapbox-gl.js'></script>
    <link href='https://api.mapbox.com/mapbox-gl-js/v3.1.2/mapbox-gl.css'
rel='stylesheet' />
    <link rel="stylesheet" href="style.css">
</head>
```

This imports the Mapbox GL JS library and Mapbox and custom CSS stylesheets into your project. Note that the script tag for libraries and frameworks are typically placed in the <head> section do they are loaded before the rest of the page content and immediately available for use.

Add the following code into the <body> tag of your HTML file:

```
<body>
    <div id="my-map" class="map-container"></div>
    <script src="script.js"></script>
</body>
```

This sets the container for the map and references the JavaScript file for map functionality. Note that scripts that interact with the content of the page (those that include event handlers, for example) are typically placed in the <body> section so that the HTML content can be fully loaded before the script is executed.

Step 2: Set up CSS file to position web map on page

In the same working directory, create a new file called *style.css* and add the following code:

```
.map-container {
    position: absolute;
    top: 0;
    bottom: 0;
    width: 100%;
}
```

This positions the map container to its closest HTML element, or in this case to the viewport as there are no other elements.

You may like to explore the W3Schools tutorial pages as a reminder of how to use CSS for styling an HTML document. Here, we are simply using the map class attribute (using the "." class selector) to set the map to fill the full width of the viewport.

Practise adjusting the width and height of the map.

Step 3: Set up JavaScript file

In the same working directory, create a new file called *script.js*

The following code snippet is provided to help you get started with writing the JavaScript code required to display a map in your web page

```
mapboxgl.accessToken = '...'; //Add default public map token from your Mapbox account

const map = new mapboxgl.Map({
    container: '...', // map container ID
    style: '...', // style URL
    center: [..., ...], // starting position [lng, lat]
    zoom: ..., // starting zoom
});
```

Where the code reads '…' update these sections. Guidance for each required update is provided below.

Mapbox tokens are available in your mapbox account under the Tokens menu. Tokens allow you to access the Mapbox API. You may use the Default public token for this exercise. However, you may like to explore the different privileges available when creating a new token.

The **mapboxgl.Map object** represents the map on the page and allows you to access associated methods and properties to change the map. Initialize the map by declaring a new map variable and set the **container** to the ID of the map div container from your html file.

For **style**, you may like to add the URL to the style you made during the exercise in Week 4 (see example style link below)

Alternatively, you may add a link to a Mapbox-owned style, e.g., 'mapbox://styles/mapbox/streets-v12'

Check the map loads in the browser as expected

Step 4: Add a data source containing GeoJSON data and visualize data as a layer

In *script.js*, add the following code

```
map.on('load', () => {

    //Add a data source containing GeoJSON data
    map.addSource('uoft-data', {
        type: 'geojson',
        data: {
            "type": "FeatureCollection",
            "features": [
                {
                    "type": "Feature",
                    "properties": {
                        "name": "Sidney Smith Hall"
                    },
                    "geometry": {
                        "coordinates": [
                            -79.39865237301687,
                            43.662343395037766
                        ],
                        "type": "Point"
                    }
                }
            ]
        }
    });

    map.addLayer({
        'id': 'uoft-pnt',
        'type': 'circle',
        'source': 'uoft-data',
        'paint': {
            'circle-radius': 6,
            'circle-color': '#B42222'
        }

    });

});
```

Take time to explore the different lines of code and add comments outlining what each section does. The [mapbox documentation] is helpful for understanding available properties and methods that are associated with an instance of a mapbox map (Instance members). For example, you may like to search the documentation for the addSource and addLayer methods.

Check the data and map load as expected.

Step 5: Add a data source from a linked GeoJSON file

Including the text of a GeoJSON file is not particularly efficient, especially when working with a large number of features or with features that have multiple nodes. Next, you will therefore learn how to add data sources from existing files and tilesets.

Download the Buildings and Toronto census tract GeoJSON files from Week 5 Class Exercise on Quercus. You may like to take time exploring the files in a browser, text editor, or GIS software.

Save the files in your working directory for the exercise.

Using GitHub Desktop, create a new repository for your working directory. Publish the repository online and unselect the option to keep code private. As your GeoJSON files are now stored within an online repository, you can use a URL to access them from within the addSource method.

In Visual Studio Code, update the *script.js* file to include the following code inside the on load event listener. The data property should point to a URL that uses the format:

`'https://raw.githubusercontent.com/yourusername/respoitoryname/main/yourfile.geojson'`

Note that if your GeoJSON is stored in a subfolder within your repository, your link will read `'.../main/yoursubfolder/yourfile.geojson'`

```
// Add a data source from a GeoJSON file
map.addSource('buildings-data', {
    type: 'geojson',
    data: '...' // Your URL to your buildings.geojson file
});

map.addLayer({
    'id': 'buildings-point',
    'type': 'circle',
    'source': 'buildings-data',
    'paint': {
        'circle-radius': 5,
        'circle-color': '#007cbf'
    }
});
```

Again, take time to make sure you understand each line of code. Adding comments is advisable and will be beneficial for when you submit your labs and assignments.

After you have published your webpage for labs or assignments using GitHub Pages, you can update your URL to the following format:

`'https://yourusername.github.io/repositoryname/yourfile.geojson'`

The Pages link is preferable to the link to raw repository files as it is based on CDN and creates less server load.

Back in your *script.js* file, it's good practice to test the URL link (e.g., using cmd key + click). If your GeoJSON opens in the browser (as plain text), the URL has been correctly configured. If you see an error, your URL may be incorrect or your repository may be set to private.

When you are happy with the URL, save any changes and view the *index.html* file in the browser. You should see additional buildings plotted as blue circles on the map.

Step 6: Add a data source from a Mapbox tileset

Using Mapbox Studio, convert the Toronto census tract GeoJSON into a vector tileset.

Using the following code snippet as a starting point, add the code into the map.on load event listener and see if you can update the code to view a polygon layer of census tracts.

```
    // Add a data source from a Mapbox tileset
    map.addSource('...', { // Create your own source ID
        'type': 'vector',
        'url': 'mapbox://username.…' // Update to your mapbox tileset ID
    });

    map.addLayer({
        'id': '...', // Create your own layer ID
        'type': 'fill', // Note this is different to point data
        'source': '...', // Must match source ID from addSource Method
        'paint': {
            'fill-color': '#888888', // Test alternative colours and style properties
            'fill-opacity': 0.4,
            'fill-outline-color': 'black'
        },
        'source-layer': '...' // Tileset NAME (diff to ID), get this from mapbox
tileset page
    },
        'uoft-buildings' // Drawing order - places layer below points
        // Here the addlayer method takes 2 arguments (the layer as an object and a
string for another layer's name). If the other layer already exists, the new layer
will be drawn before that one
    );
```

Save your edits and update your repository.