Maps and you!

# Districtmap.ejs

The code begins with the usual HTML document structure, including the <html>, <head>, and <body> tags.

1. Inside the <head> section, there are various elements:
   * An inclusion of a partial file named \_header.
   * A link to an external CSS file named styles.css.
   * Meta tags for defining the viewport.
   * A <title> element specifying the title of the web page.
   * A link to the Leaflet CSS file (a library for interactive maps).
   * A script tag including the Leaflet JavaScript file.
2. Inside the <body> section:
   * There's a <div> element with an id attribute set to "map". This div is where the map will be displayed.
   * The JavaScript code begins with the initialization of a Leaflet map, setting the initial view and zoom level.
   * A tile layer from OpenStreetMap is added to the map using the L.tileLayer function.
   * An attribution tile layer from OpenStreetMap is added to the map to provide credit and a link to the source.
   * A marker is added to the map representing the "York County School of Technology" and displays a popup with the same information.
   * GeoJSON layers are added to the map for each participating district. The GeoJSON data for each district is parsed from the server-side code using the JSON.parse function and added to the map using the L.geoJSON function.
   * Each GeoJSON layer is styled with a specific color and weight for the outline.
   * Popups are bound to specific GeoJSON layers to display the district name.
3. Lastly, there are inclusion of partial files in the <%- include('partials/\_footer') -%> format. These partials likely contain footer content for the web page.

Please note that this code assumes the availability of GeoJSON data for each district (geojson1, geojson2, ..., geojson8). Without the actual data, the map won't display the boundaries of the districts. Additionally, the code relies on external dependencies such as Leaflet and the specified CSS file, so ensure that those resources are properly linked and accessible for the map to function correctly.

# App.js map section

The provided code is a JavaScript function that fetches GeoJSON data for a given OpenStreetMap (OSM) relation ID using the Overpass API and the osmtogeojson library. Here's a breakdown of the code:

1. The code begins by importing the osmtogeojson library and the https module, which is a built-in Node.js module for making HTTPS requests.
2. The fetchGeoJson function is defined, which takes a parameter relationId representing the OSM relation ID for which the GeoJSON data should be fetched.
3. Inside the function, an Overpass query is constructed using the relation keyword and the provided relationId. The query requests the JSON format and includes the out geom statement to retrieve the geometries of the relation.
4. The query is URL-encoded and combined with the Overpass API base URL (https://overpass-api.de/api/interpreter?data=) to create the full URL.
5. The function returns a Promise that wraps the asynchronous operation of making an HTTPS request and processing the response.
6. Within the Promise constructor, an HTTPS request is made to the Overpass API using the https.get function. The URL is passed as an argument, and a callback function is defined to handle the response.
7. Inside the response callback, a variable named data is initialized to an empty string. This variable will accumulate the response data as it is received.
8. The response object emits a 'data' event whenever a chunk of data is received. The callback for this event appends the received chunk to the data variable.
9. The response object also emits an 'end' event when all the data has been received. In the 'end' event callback, the received data is parsed as JSON using JSON.parse. Then, the osmtogeojson function is called with the parsed data to convert it into GeoJSON format.
10. If the conversion is successful, the Promise is resolved with the GeoJSON data using the resolve function. If any errors occur during the process, the Promise is rejected with the error using the reject function.
11. Additionally, an 'error' event listener is attached to the HTTPS request to handle any errors that occur during the request.

To use this code, you can call the fetchGeoJson function and provide it with the desired OSM relation ID. The function returns a Promise that resolves to the fetched GeoJSON data when the request is successful or rejects with an error if an error occurs during the process.