Assignment 4. Creating a Fashion Weather App Utilizing Streaming Open Data

Due November 15th, 2023 @ 11:59 PM Worth 25 points

In teams, you will build an app that interprets various streams of live weather data and suggest how to dress and what to bring to deal with current conditions. The user will click on a map, indicating their location, and the application will capture these coordinates. The coordinates will be passed to a weather service API, which will return various streaming/live data such as temperature, precipitation, air quality, and pollen count. Finally, the app will interpret these data and compile plain text recommendations that inform the user of what they need to be comfortable outside in such conditions.

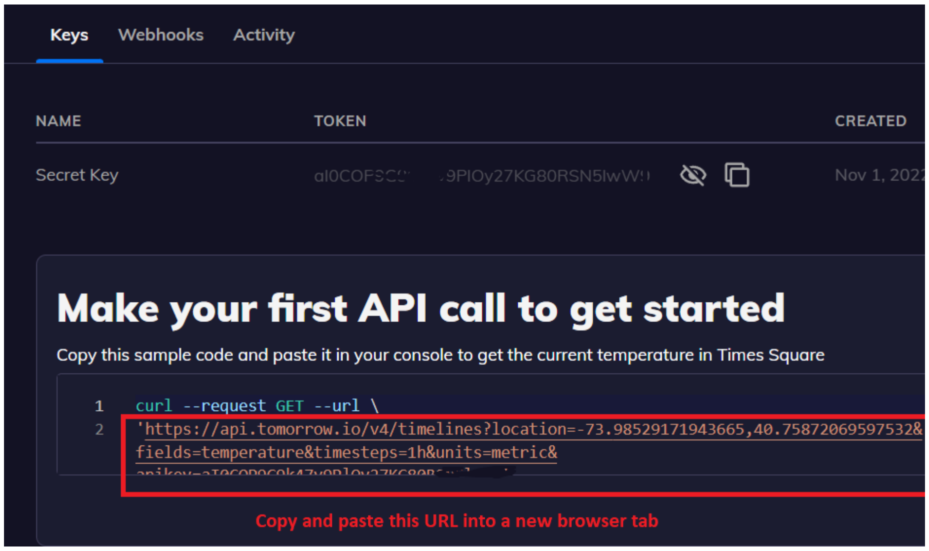
Goals

After you've completed this assignment, you should be better able to:

- Listen for DOM events to pass information to you code (i.e., clicking the page)
- Connect to an API
- Use Javascript to do API "calls" and parse returned JSON data
- Create a response matrix/decision tree that translates quantitative data points into actionable qualitative observations (Basically what an AI does but at a much higher level!)
- Finally, gain experience in working with streaming data.

Tasks

1. This assignment requires that you use the Tomorrow.io (https://www.tomorrow.io/) weather API. One member of your team will need to create an account (https://app.tomorrow.io/signup). Please use your McGill email and answer any questions you may be asked to affirm that you are a student, using it for educational purposes.
2. Create an API key to use for your application (https://app.tomorrow.io/development/keys). On this page you will also find a URL that is ready for you to test. Copy and paste the URL into your browser and see what you get in return. The JSON data you see is what you are eventually loading into your script.



Keys Webhooks Activity

NAME	TOKEN	CREATED
Secret Key	aI0COF[redacted].9PIOy27KG80RSN5lwW[redacted]	Nov 1, 202

Make your first API call to get started

Copy this sample code and paste it in your console to get the current temperature in Times Square

```
curl --request GET --url \
'https://api.tomorrow.io/v4/timelines?location=-73.98529171943665,40.75872069597532&
fields=temperature&timesteps=1h&units=metric&
apikey=[redacted]
```

Copy and paste this URL into a new browser tab

a. Look at the structure of this URL. It has parameters that can be changed to return different information. By default, this URL has “timesteps=1h” if we change this to “timesteps=current” we get the current (live) temperature.

b. **Note: At the time of writing this assignment, we have determined that the latitude and longitude coordinates on the default example are reversed. This is common in apps and always must be checked. Change around (“pivot”) the numbers to get an accurate location.[1] Please make sure you always have LATITUDE first followed by LONGITUDE.**

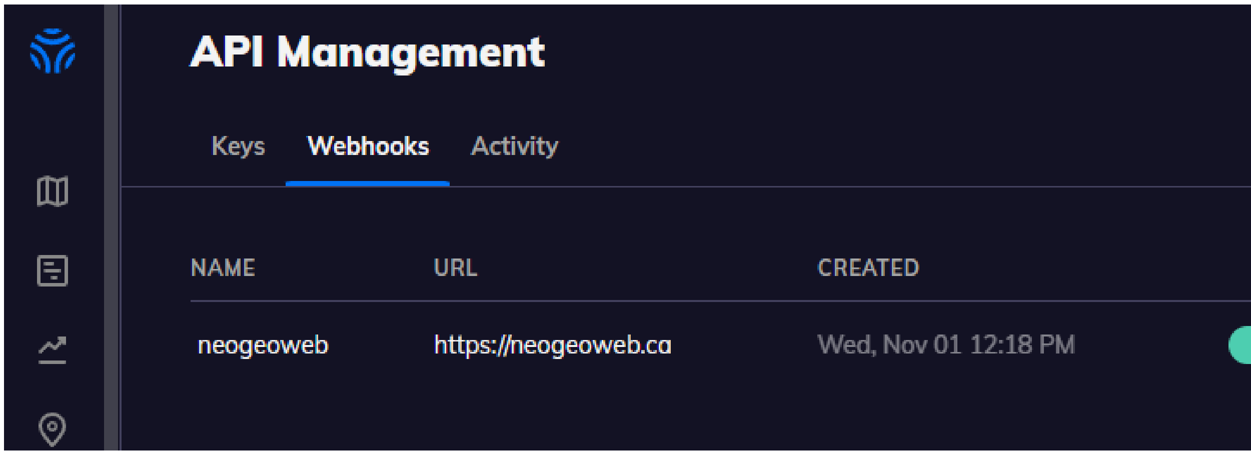
As provided from the website the default the example url contained:

“location=-73.98529171943665,40.75872069597532,”

Whereas, it should have been:

“location=40.75872069597532,-73.98529171943665”

c. Before you leave this page, be sure to go to the ‘webhooks’ tab and add https://neogeoweb.ca as an authorized url



API Management

Keys Webhooks Activity

NAME	URL	CREATED
neogeoweb	https://neogeoweb.ca	Wed, Nov 01 12:18 PM

[1] Accuracy’s relative, right?

3. Develop a web-based app (HTML pages) that translates Tomorrow.io data into practical, actionable suggestions for a user. You may say on your home page, "Our app focuses on European weather conditions. The map will load with Europe in focus and customize the style to have any relevant labels and country boundaries visible. The point file we load will feature European capitals. Our data requests will focus on temperature, humidity, precipitation, cloud cover and wind gust speed. We will suggest the correct kinds of outerwear (e.g., Jacket, windbreakers, none) for Chesterton (i.e., the nearest city/town), whether or not the user needs an umbrella/galoshes,[2] should they should put on sunscreen and whether or not they can expect it to be a humid or windy day.
    a. To maximize the utility of your application you need to start with an embedded world map from Mapbox GL JS. Choose an area on which to focus. You are given free rein to zoom into any area of the world and style the map as you desire, but you should establish a clear use case and style and stick to that (see example).

       Remember that weather conditions may vary by your chosen area so you will need to customize your fashion advice (i.e., no -40° C coats for Egypt right now).

       Keep in mind: The expected behavior for this API is that when you submit any reasonable latitude or longitude (within the bounds of where there are instruments that measure weather) you will get some data returned to you.[3]

    b. Find a dataset of latitude and longitude coordinates for relevant cities/towns within the area of your app and add this as a point layer to Mapbox (minimum requirement is 50 points).

    c. Make sure to have a title for your applications and some relevant explanatory text in the HTML body of your app.

4. Write JS code that waits for and then captures the coordinates of where a user clicks on the map.
    a. Add this click to the map as a Mapbox GL JS point (drop a pin). Your code should allow for the user to click somewhere else on the map and drop a new pin, removing the old one.

    b. Snap your pin to the nearest city/town point. That will be the location you send to Tomorrow.io and be included in the information you send to the end user.

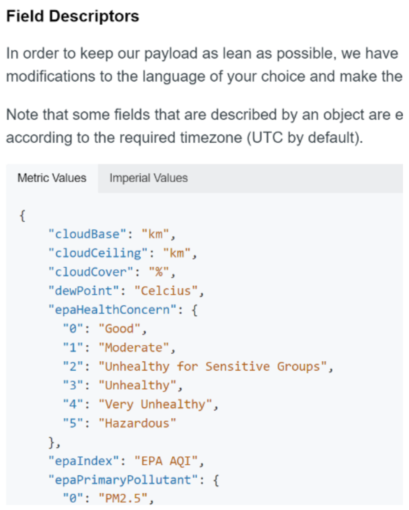
5. Use the coordinate information to construct the GET request URL that you will send to the Tomorrow.io API. The request will follow the format you saw in Step 2. **'var url = https://api.tomorrow.io/v4/timelines?location=x,y …**

[2] How many of you needed to look up that word?

[3] Some areas have more data than others—not sure how much data is available at the poles. Also you'll want a location that has some temporal variability in weather conditions.

a. You will need to ask (deploy a 'get') the API for minimum 5 different kinds of weather data. This will require some trial and error to understand the kind of data you get back for each kind of weather information (i.e., temperature will return integers of degrees; pollen count will return 0-5 values that represent various conditions.)

b. Parameters have different units and they're updated at different intervals. All the parameters that you can include in your GET request here: https://docs.tomorrow.io/reference/weather-data-layers



Field Descriptors

In order to keep our payload as lean as possible, we have
modifications to the language of your choice and make the

Note that some fields that are described by an object are e
according to the required timezone (UTC by default).

```
{
    "cloudBase": "km",
    "cloudCeiling": "km",
    "cloudCover": "%",
    "dewPoint": "Celcius",
    "epaHealthConcern": {
      "0": "Good",
      "1": "Moderate",
      "2": "Unhealthy for Sensitive Groups",
      "3": "Unhealthy",
      "4": "Very Unhealthy",
      "5": "Hazardous"
    },
    "epaIndex": "EPA AQI",
    "epaPrimaryPollutant": {
      "0": "PM2.5",
```

c. Pay special attention on the same page to the section called "field descriptions". Here you will see all the possible datasets that you can ask the Tomorrow.io API for and the data that you will get in return. Try replacing the "fields=temperature" section of the sample URL you saw in Step 2 (e.g., "fields=cloudCover").

d. Many times requests are embedded in the URL. You should concatenate your request into a single URL you send to the Tomorrow.io API. Add the snapped user x,y into the location section and include all the comma separated datasets into the "fields=temperature,cloudCover … " section.

6. Import the JSON response you get from the Tomorrow.io API into your code. You will need to add the JQuery library to your HTML.[4] Adding a library expands the functionality of your code. Libraries are added in the head section of your code, even as you use the functions in the script section
   a. In the <head> section of your .html page add the following script tag:

      ```
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
      ```

   b. Use the getJSON() function, supplied by jquery, to send your GET request URL to the Tomorrow.io API . This function stores the response as a JSON in an object variable.
      i. Your code must not call the getJSON() function until you have captured the user input and added the x,y to the GET request URL. One thing must trigger the other to happen. Pay attention to what is in scope and when.

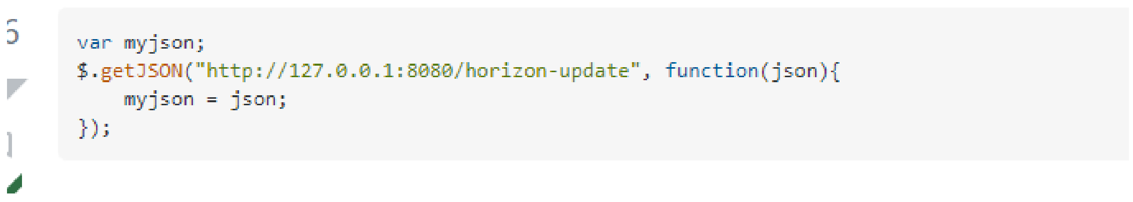
   c. You also will be working with "callbacks", which look straightforward but are beyond the ability of most students in class. For instance, sometimes your API call will take longer to return your JSON file than it takes to execute the rest of the code in your <script> section. In this instance your code conducts its analysis on a null value and fails to work.

[4] Have a look at what JQuery is here: https://www.w3schools.com/jquery/jquery_intro.asp

i. To combat this, you need to put all your analysis inside the getJSON function. This is the callback in JS. Here one parameter in the call is actually a function.

ii. In the example in 6c, this would mean replacing the “myjson = json” line with all your analysis code or calls to functions written outside this section.

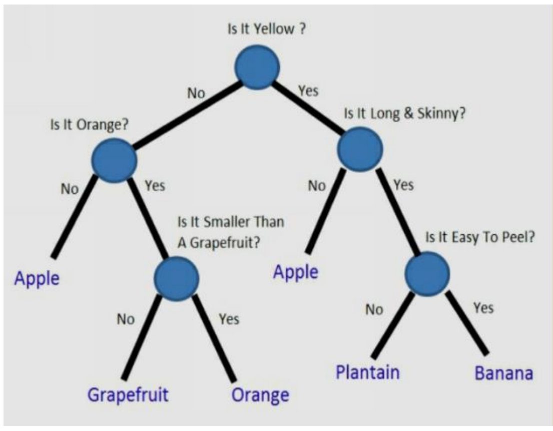
iii. Take a look at how this works on StackOverflow https://stackoverflow.com/questions/15764844/jquery-getjson-save-result-into-variable, specifically this answer:



```
var myjson;
$.getJSON("http://127.0.0.1:8080/horizon-update", function(json){
    myjson = json;
});
```

iv. Note that the JQuery syntax is slightly different than Javascipt. When you are using a method from the JQuery library it is written beginning with “$.” https://www.w3schools.com/jquery/jquery_intro.asp to know more

7. The final step for your app is to return fashion advice to your user. This advice should include output like what they need to bring with them; what they need to wear; and what to expect when they go outside. For this step we want you to think like an AI programmer. Write a decision tree, which can be used by algorithms referred to as Explainable AI. Here we’re employing it for GeoAI. “GeoXAI is “a set of approaches (from plain text and visualizations to algorithms) that integrate geographic structures and knowledge to improve the understanding of the geographic implications of AI for various audiences.” (Xing and Sieber, 2023, p 627). Decision trees can be easily read and even mimic a human approach to decision making by breaking the choice into smaller sub-choices.”. [5]



a. This will require some critical thinking and creativity on your part. Think about how you take the quantitative data and translate it into some plain text advice. You may offer whatever practical advice you can think of. We are not expecting you to be an expert here or to do any scientific research, but you do need to use common sense (e.g., cold temperature = coat). The more effort you put into the section, the better the advice for your user, and the better your evaluation for the assignment.

[5] Xing, J. and Sieber, R. 2023. The challenges of integrating explainable artificial intelligence into GeoAI. *Transactions in GIS.*
and
https://towardsdatascience.com/explainable-ai-xai-with-a-decision-tree-960d60b240bd?

b. XAI depends on the user; most often it's computer scientist. So you'll need to consider who your target audience is.[6]

c. Your application's decision tree will be instantiated in code. It should use conditional statements and string concatenation to create a single block of text that you display somewhere on your page or as an alert for the user.

d. Do not forget to mention the nearest location to the location the user selected on the map.
    i. "You clicked on Manchester, UK. Current weather conditions are…."

8. As usual, document your code. Also structure your code. When you work with features like user clicks and URLs represented by strings, these should be stored as global variables. As much as possible you should structure your code with functions.

9. Write a report.
    a. Your group report should:
        i. Describe the purpose of your app.
        ii. Justify the style choices and map options you chose. Justify your choice of location.
        iii. Document the structure of your app. Build a decision tree model (think of it as a cartographic model) to explain the step-by-step "XAI" logic of your app.
        iv. Create a response matrix that translates your "received" various weather observations at the specified x,y,t into qualitative actions (i.e., the fashion advice for any particular observation)
    b. For each of you in the team, reflect on the assignment (approximately one page each):
        i. Find two-three short articles or posts on streaming data (cite as well). What are their main points? Anything we missed in class? Coordinate with others in class so you're not relying on the same posts.
        ii. What information sources did you use to complete your portion of the assignment? For example, did you view any Youtube videos? Which ones? Did you utilize stackoverflow? How?
        iii. Did you use a GPT? For what purpose?
        iv. Make sure that all the individual reflections for your team are appended to the bottom of your report.

[6] Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. 2020. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, 58, 82–115.

Notes:

- You can do development on your local machine but to test your getJSON() requests you must publish your HTML to the server and run the webpage from there, which is why we add https://neogeoweb.ca as an authorized site in Step 2c.
- You will probably find Tomorrow.io rejects any getJSON() request they receive if you try to load the page saved to your local machine.

Submission:

1) Email the code, the URL to your working app, and your report describing the process (including individual reflections) to renee.sieber@mcgill.ca and sichen.wan@mail.mcgill.ca.