Task 3

Display the User’s Public IP

Using the Fetch API

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global `fetch()` method that provides an easy, logical way to fetch resources asynchronously across the network.

This kind of functionality was previously achieved using `XMLHttpRequest`. Fetch provides a better alternative that can be easily used by other technologies such as `Service Workers`. Fetch also provides a single logical place to define other HTTP-related concepts such as CORS and extensions to HTTP.

The `fetch` specification differs from `jQuery.ajax()` in the following significant ways:

- The Promise returned from `fetch()` **won't reject on HTTP error status** even if the response is an HTTP 404 or 500. Instead, as soon as the server responds with headers, the Promise will resolve normally (with the `ok` property of the response set to false if the response isn't in the range 200–299), and it will only reject on network failure or if anything prevented the request from completing.
- Unless `fetch()` is called with the `credentials` option set to `include`, `fetch()`:
  - won't send cookies in cross-origin requests
  - won't set any cookies sent back in cross-origin responses
  - As of August 2018, the default credentials policy changed to same-origin. Firefox was also modified in version 61.0b13)

A basic fetch request is really simple to set up. Have a look at the following code:

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```
Copy to Clipboard

Here we are fetching a JSON file across the network and printing it to the console. The simplest use of `fetch()` takes one argument — the path to the resource you want to fetch — and does not directly return the JSON response body but instead returns a promise that resolves with a `Response` object.

The `Response` object, in turn, does not directly contain the actual JSON response body but is instead a representation of the entire HTTP response. So, to extract the JSON body content from the `Response` object, we use the `json()` method, which returns a second promise that resolves with the result of parsing the response body text as JSON.