

## 6.1.4 Create a New JavaScript File to Request Data

Amiko has given us a lot of starter code, but we don't need to do anything with it at the moment. Instead, we'll focus our efforts on sorting out the JavaScript logic.

Create a new JavaScript file located at `assets/js/homepage.js`. In `index.html`, add a `<script>` element to link to the new JavaScript file. It should look like the following:

```
<!-- add script at the bottom of the body -->
<script src="./assets/js/homepage.js"></script>
</body>

</html>
```

In `homepage.js`, add the following function and function call:

```
var getUserRepos = function() {
  console.log("function was called");
};

getUserRepos();
```

Now open and test the app in the browser. Even though you won't see much yet, verifying the console log will at least ensure that the JavaScript file is linked correctly.

Note that the function is named `getUserRepos()`. Fittingly, this page searches for GitHub users and lists all of their repositories, tallying the open issues for each one. How do you get this information from GitHub, though? As mentioned earlier, it wouldn't work to copy and paste the data, because it's so changeable—a repo's issue count could change the same day that you copy its information.

Thankfully, GitHub provides a **server-side API** that allows apps like yours to request data as needed. You've already used web APIs, like the Storage API and Console API, which provided an interface to use browser features like `localStorage` and `console.log()` statements. Similarly, server-side APIs provide an interface to access data from a server.

What exactly is a server, though? A **server** is a piece of hardware set up to provide resources to other devices (often called **clients**). Clients can send requests to these servers in a number of ways. For example, a common HTTP (or HTTPS) request is to visit a URL in the browser (e.g., `https://www.google.com`). In turn, the server will send back the resource it defined for that URL, be it an HTML file, a PDF, an image, or data.

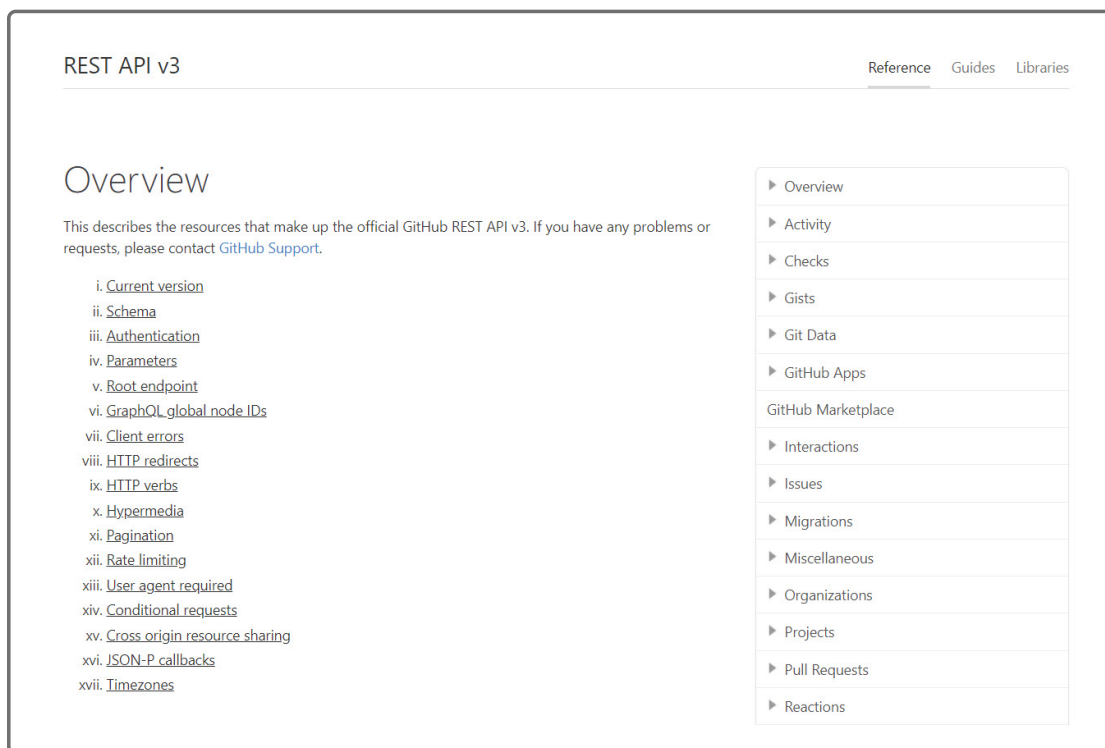
If an application has a database, that database lives on the server. Browsers and clients can't talk directly to a database, but they can make requests to the server that controls the database. Thus, the server can read and format the data from the database and send it back to the client.

For a more in-depth look at servers and clients, watch the following video:



Consider a website like Facebook or Twitter, where new posts, pictures, and comments load into view as the user scrolls down the page. To retrieve that data as needed, these websites make requests to their own servers via server-side APIs. Some companies make their server-side APIs public; other websites can then use those public APIs to make the same requests and load third-party data into their own apps.

This brings us back to GitHub, which is one such company with a public API. Visit the [GitHub API documentation](https://developer.github.com/v3/) [\(https://developer.github.com/v3/\)](https://developer.github.com/v3/) and skim over some of the information on the homepage. The page includes an overview of how to use the API and a list of data features on the right, as seen in the following image:

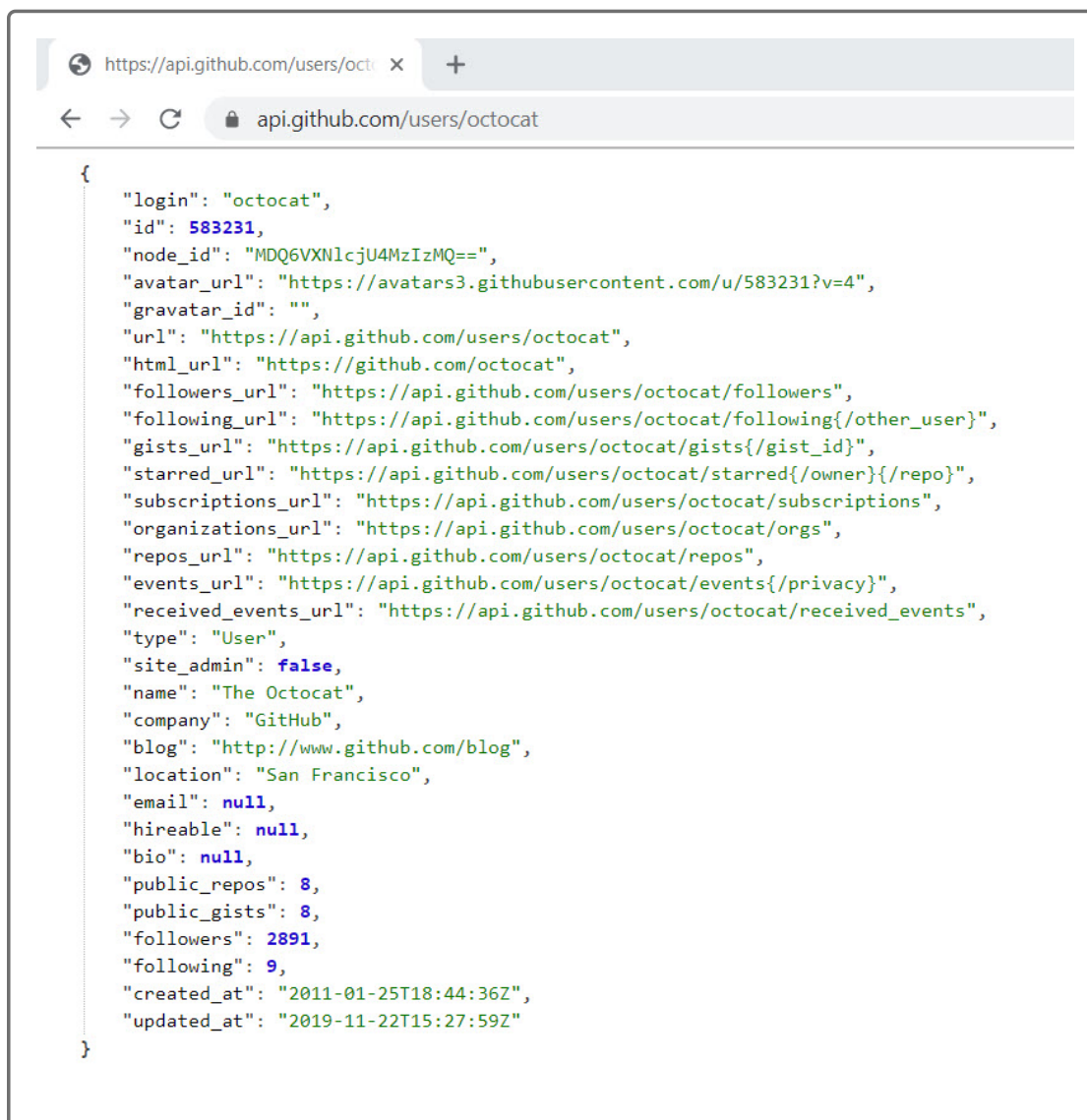


Using their public API, you can request almost any data about accounts on GitHub, like their repositories, contributors, commit histories, and more.

To obtain this information, you can make a request to one of GitHub's API **endpoints**. An endpoint specifies which resources you can access. All GitHub API access is over HTTPS, building off the base URL

<https://api.github.com>. For example, the following URL returns data about a user named Octocat: <https://api.github.com/users/octocat>.

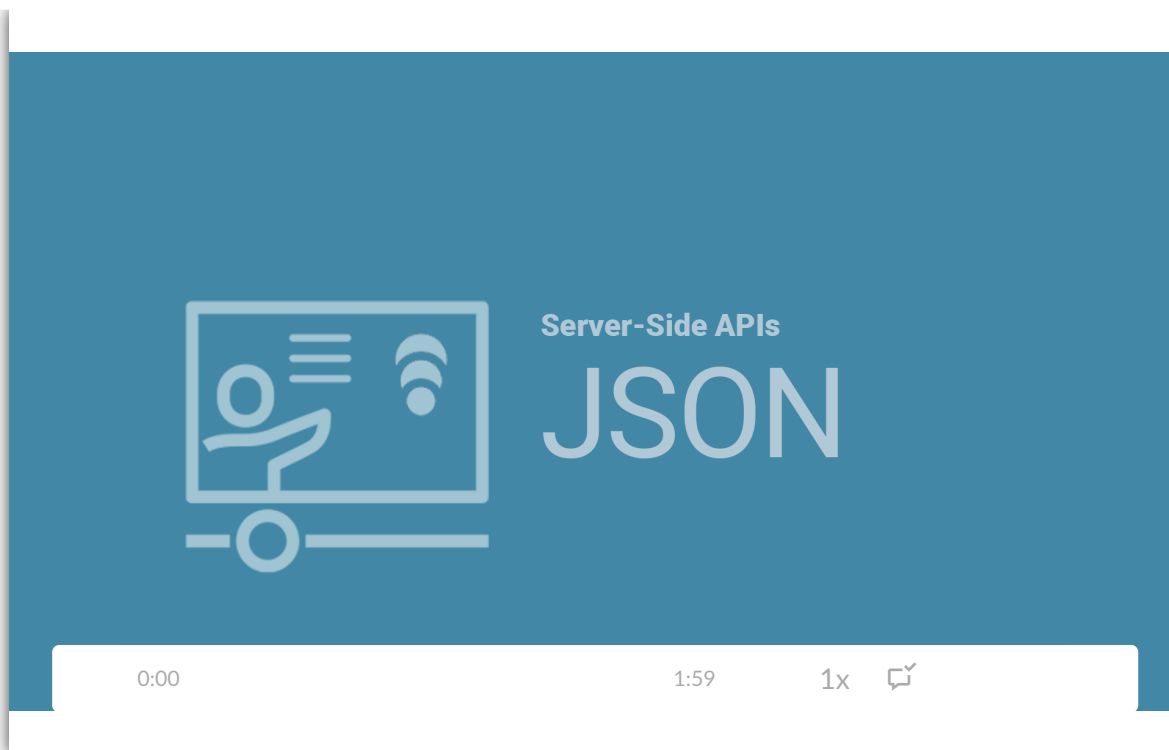
Open the [users/octocat](#) URL in a new tab in Chrome. If you have the [JSON Formatter Chrome extension](#) (<https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa?hl=en>) installed, it should look like the following image:



```
{
  "login": "octocat",
  "id": 583231,
  "node_id": "MDQ6VXNlcjU4MzIzMQ==",
  "avatar_url": "https://avatars3.githubusercontent.com/u/583231?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/octocat",
  "html_url": "https://github.com/octocat",
  "followers_url": "https://api.github.com/users/octocat/followers",
  "following_url": "https://api.github.com/users/octocat/following{/other_user}",
  "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/octocat/starred{/owner}/{/repo}",
  "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
  "organizations_url": "https://api.github.com/users/octocat/orgs",
  "repos_url": "https://api.github.com/users/octocat/repos",
  "events_url": "https://api.github.com/users/octocat/events{/privacy}",
  "received_events_url": "https://api.github.com/users/octocat/received_events",
  "type": "User",
  "site_admin": false,
  "name": "The Octocat",
  "company": "GitHub",
  "blog": "http://www.github.com/blog",
  "location": "San Francisco",
  "email": null,
  "hireable": null,
  "bio": null,
  "public_repos": 8,
  "public_gists": 8,
  "followers": 2891,
  "following": 9,
  "created_at": "2011-01-25T18:44:36Z",
  "updated_at": "2019-11-22T15:27:59Z"
}
```

You might notice that the `users/octocat` URL displays a JavaScript object instead of HTML content. To be more specific, data displayed in this format is called **JavaScript Object Notation**, or **JSON**. APIs commonly use JSON to deliver data because developers can easily parse it for the information they want.

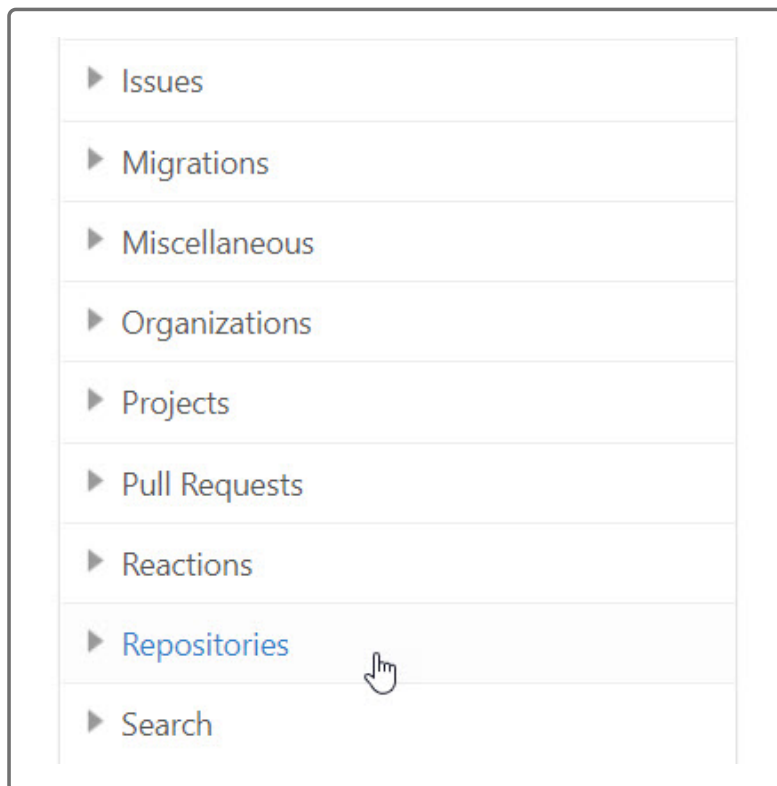
For more information about the JSON format, watch the following video:



Look at the JSON from `users/octocat` again. If we assigned this object to a variable called `response`, how would we access the user's location and number of followers? It would look something like this:

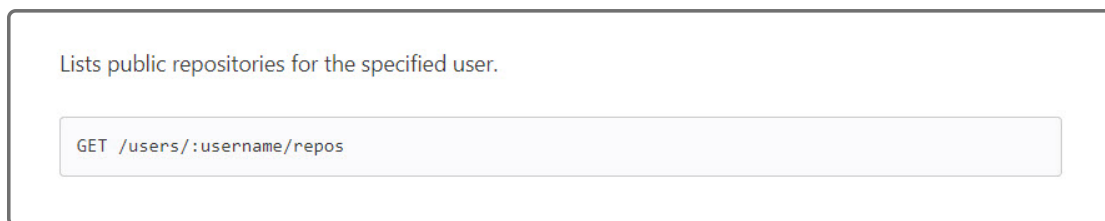
```
console.log(response.location);  
console.log(response.followers);
```

Unfortunately, in the context of the Git it Done app, this `users/<username>` endpoint doesn't help us much, because it doesn't provide any repository information. However, GitHub offers many endpoints beyond user data. Look again at the righthand side of their documentation website for a list of other options, as seen in this image:



Visit the [GitHub API documentation for repositories](https://developer.github.com/v3/repos/) (<https://developer.github.com/v3/repos/>), which claims to return a list of a user's repositories. Now we're getting somewhere! This data is exactly what we need for the app, but how do we import it?

If you scroll through the repositories documentation, you'll see the following snippet:



Unfortunately, that doesn't tell us much about how to use this endpoint. Let's backtrack to the [GitHub API homepage](https://developer.github.com/v3/) (<https://developer.github.com/v3/>). This page offers a couple of helpful snippets, including the following:

All API access is over HTTPS, and accessed from `https://api.github.com`. All data is sent and received as JSON.

```
curl -i https://api.github.com/users/octocat/orgs
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 12 Oct 2012 23:33:14 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Status: 200 OK
ETag: "a00049ba79152d03380c34652f2cb612"
X-GitHub-Media-Type: github.v3
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4987
X-RateLimit-Reset: 1350085394
Content-Length: 5
Cache-Control: max-age=0, private, must-revalidate
X-Content-Type-Options: nosniff
```

Well, we could certainly try this `curl` thing.

On your computer, open a new terminal window and run the following command:

```
curl https://api.github.com/users/octocat/repos
```

This will print the following:



```
$ curl https://api.github.com/users/octocat/repos
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           0         0     0         0          0      0      0      0
0         0     0     0         0          0      0      0      0
{
  "id": 132935648,
  "node_id": "MDEwO1JlcG9zaXRvcnkxMzI5MzU2NDg=",
  "name": "boysenberry-repo-1",
  "full_name": "octocat/boysenberry-repo-1",
  "private": false,
  "owner": {
    "login": "octocat",
    "id": 583231,
    "node_id": "MDQ6VXN1cjlU4MzIzMQ==",
    "avatar_url": "https://avatars3.githubusercontent.com/u/583231?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/octocat",
    "html_url": "https://github.com/octocat",
    "followers_url": "https://api.github.com/users/octocat/followers",
    "following_url": "https://api.github.com/users/octocat/following{/other_user}",
    "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/octocat/starred{/owner}/{/repo}",
    "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
    "organizations_url": "https://api.github.com/users/octocat/orgs",
    "repos_url": "https://api.github.com/users/octocat/repos",
    "events_url": "https://api.github.com/users/octocat/events{/privacy}",
    "received_events_url": "https://api.github.com/users/octocat/received_events",
    "type": "User",
    "site_admin": false
  },
}
```

Congratulations, you've made your first programmatic HTTP request, and in doing so, you verified an endpoint! Using `curl`, you contacted the GitHub API endpoint, and it returned a bunch of JSON data. Now you know that this endpoint works, but how does this help you with the actual web application? Why use `curl` if it only works in the terminal?

API providers like GitHub tend to offer `curl` as an example in their documentation because they can't predict what language you'll use to write your app. It's simpler for them to provide a single command rather than providing examples for JavaScript, C#, PHP, Python, Ruby, and so on—and they assume developers can work out the language specifics after verifying the endpoint.

## DEEP DIVE ▲

### DEEP DIVE

The `curl` command is part of a bigger project called cURL that's been around since 1997. It is another open source library whose [source code is hosted on GitHub \(https://github.com/curl/curl\)](https://github.com/curl/curl)!

cURL offers other capabilities besides just verifying API endpoints. For example, the following command would save the data returned from the `/users/octocat` endpoint to a local file called `octocat.json`:

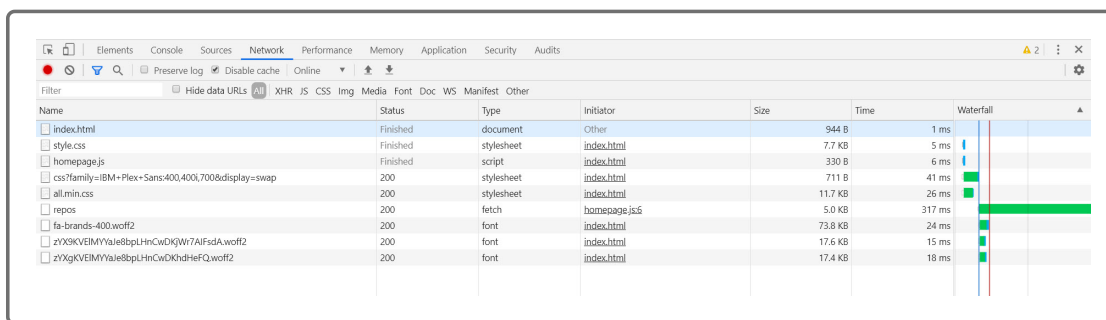
```
curl -o octocat.json https://api.github.com/users/octo
```

If we can only use `curl` in the terminal, what options do we have for working in the browser? Fortunately, browsers provide other web APIs designed to communicate with server-side APIs. The newest of such APIs is called the **Fetch API**, because it fetches resources.

Let's jump right into a practical example. Return to `homepage.js` and update the `getUserRepos()` function as follows:

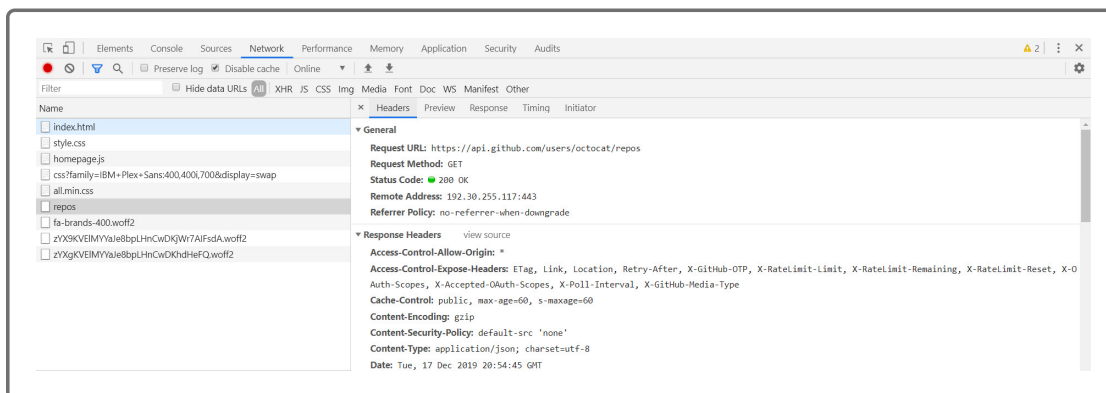
```
var getUserRepos = function() {  
  fetch("https://api.github.com/users/octocat/repos");  
};
```

Refresh the page in the browser. You won't notice any changes yet in the UI, but the browser did make an HTTP request to `/users/octocat/repos` on your behalf. To see this, open the Chrome DevTools and navigate to the Network tab. With the Network tab open, refresh the page again. You should now see something like the following image:



This is a list of all the resources the browser requested when trying to load your webpage. HTML, CSS, and JS files, as well as any third-party Google fonts, are all resources that have to be requested in order for your page to load. Try clicking on the Type column header to group similar request types together.

You'll notice one request type listed as "fetch," representing the request made to the GitHub API. Click on the name of the request to open a panel with more information. You should see a screen that looks like this image, if the "Headers" tab is selected:



Yikes, there's a lot of potentially confusing stuff in there. While most of it won't apply to us, note the Remote Address, which has a value along the lines of `192.30.255.117`. This happens to be the **IP address** for GitHub's API. Every computer or server connected to the internet has an IP address to uniquely identify it. But thanks to the Domain Naming System (or DNS), you can use a more human-readable hostname like `api.github.com`, and the DNS will translate it to the correct IP address on your behalf.

## DEEP DIVE ▲

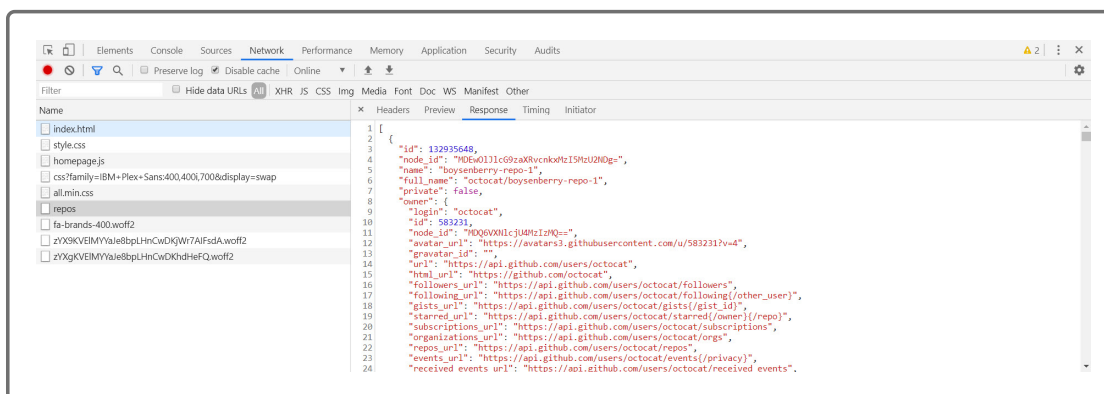
## DEEP DIVE

You can also see this IP address and hostname relationship by using the `ping` command. Open a terminal window again and run the following command:

```
ping api.github.com
```

You'll see that the replies come from an IP address such as `192.30.255.117`. Like the `curl` command, you can use `ping` to verify an API endpoint, but it's used more to test the connection itself than to see the returned data.

With the DevTools Network tab still open, click the Response tab in the info panel. It should look like this image:



The response is the data that the request returned. Note that the data is formatted as JSON consisting of an array of objects. It would certainly be great to have access to that array in JavaScript!

We'll get there soon enough. But first, make sure you use Git to add, commit, and push the current progress to GitHub.

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.