

## 6.2.5 Display Response Data on Page

Remember, web developers often have to capture a user interaction, form an HTTP request from that interaction, and display the response data on the page. We've done two of those three tasks so far: we've captured a user interaction and used it to form an HTTP request. Now we need to display the important parts of the response data on the page.

But first, you should know that you may not need all of the response data you receive. A lot of HTTP requests to server-side APIs receive responses with way more JSON data than necessary. The amount of data you need from an HTTP response will vary from project to project, based on the API you make a request to and the type of app you're building, but in this case you just need to know the username, the repository name, and how many issues the repository has.

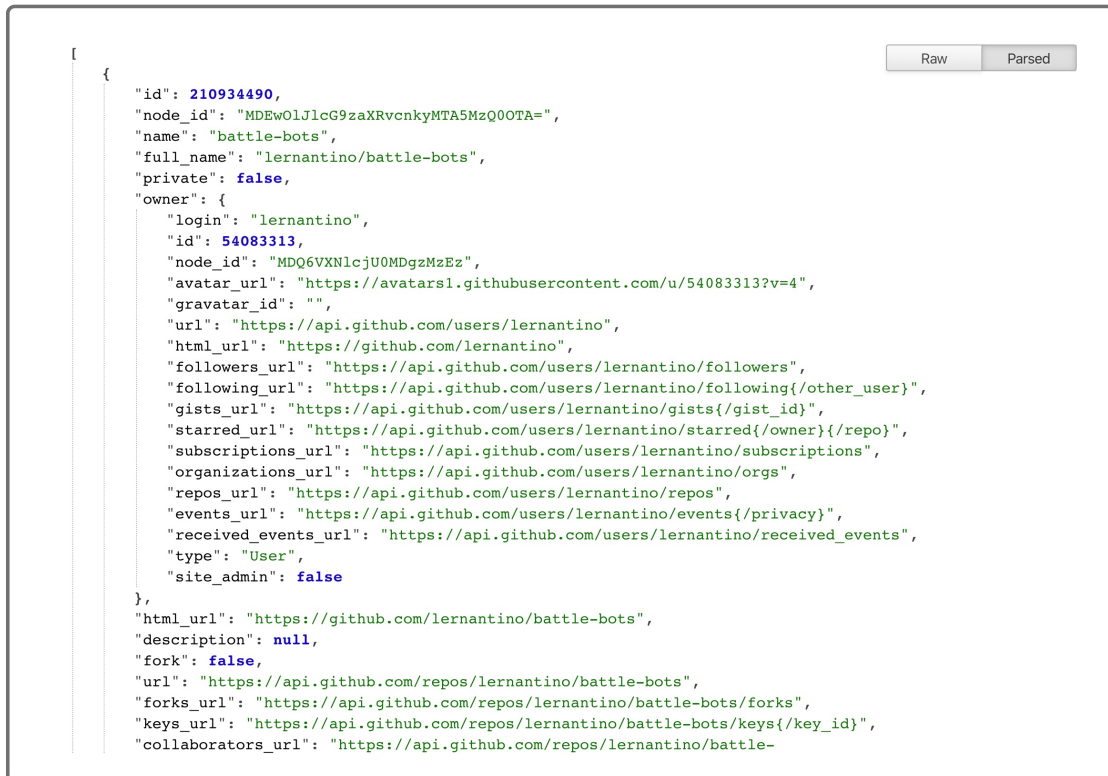
---

### Study the Response Data

Numerous tools can help us find the data we need, but the easiest way to study the response of a simple API request like this one is to use the browser! Let's study the response data so that we can identify the properties we want to use. Enter the following into the browser's URL bar:

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

When you navigate to that URL, you'll see the same response as the `fetch()` response, like this image shows:



```
[
  {
    "id": 210934490,
    "node_id": "MDEwOlJlcG9zaXRvcnkyMTA5MzQ0OTA=",
    "name": "battle-bots",
    "full_name": "lernantino/battle-bots",
    "private": false,
    "owner": {
      "login": "lernantino",
      "id": 54083313,
      "node_id": "MDQ6VXNlcjU0MDgzMzEz",
      "avatar_url": "https://avatars1.githubusercontent.com/u/54083313?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/lernantino",
      "html_url": "https://github.com/lernantino",
      "followers_url": "https://api.github.com/users/lernantino/followers",
      "following_url": "https://api.github.com/users/lernantino/following{/other_user}",
      "gists_url": "https://api.github.com/users/lernantino/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/lernantino/starred{/owner}/{/repo}",
      "subscriptions_url": "https://api.github.com/users/lernantino/subscriptions",
      "organizations_url": "https://api.github.com/users/lernantino/orgs",
      "repos_url": "https://api.github.com/users/lernantino/repos",
      "events_url": "https://api.github.com/users/lernantino/events{/privacy}",
      "received_events_url": "https://api.github.com/users/lernantino/received_events",
      "type": "User",
      "site_admin": false
    },
    "html_url": "https://github.com/lernantino/battle-bots",
    "description": null,
    "fork": false,
    "url": "https://api.github.com/repos/lernantino/battle-bots",
    "forks_url": "https://api.github.com/repos/lernantino/battle-bots/forks",
    "keys_url": "https://api.github.com/repos/lernantino/battle-bots/keys{/key_id}",
    "collaborators_url": "https://api.github.com/repos/lernantino/battle-
```

Keep in mind, the response itself is an array of objects, with each object holding one repository's data. The nice thing about working with data structured as JSON is that the property names are the same for each repository, so we only need to study this first one.

## HIDE PRO TIP

You can also study this data in the Chrome DevTools Network tab! After you click on a network request, you can open the submenu labeled Preview to see the response data formatted as JSON.

Let's identify how the details we need are labeled in the response data. We're looking for the `name` property (for the repository name), the `open_issues_count` property (for the number of issues), and the `login` property in the nested `owner` object (for the repository owner).

We've identified the data now, so we can work on getting it to the page! This'll involve a `for` loop and a fair amount of DOM functionality, so let's create a new function for it.

## Create a Function to Display Repos

Let's start by creating a new function called `displayRepos()`. This function will accept both the array of repository data and the term we searched for as parameters. Add the following code to `homepage.js`:

```
var displayRepos = function(repos, searchTerm) {  
  console.log(repos);  
  console.log(searchTerm);  
};
```

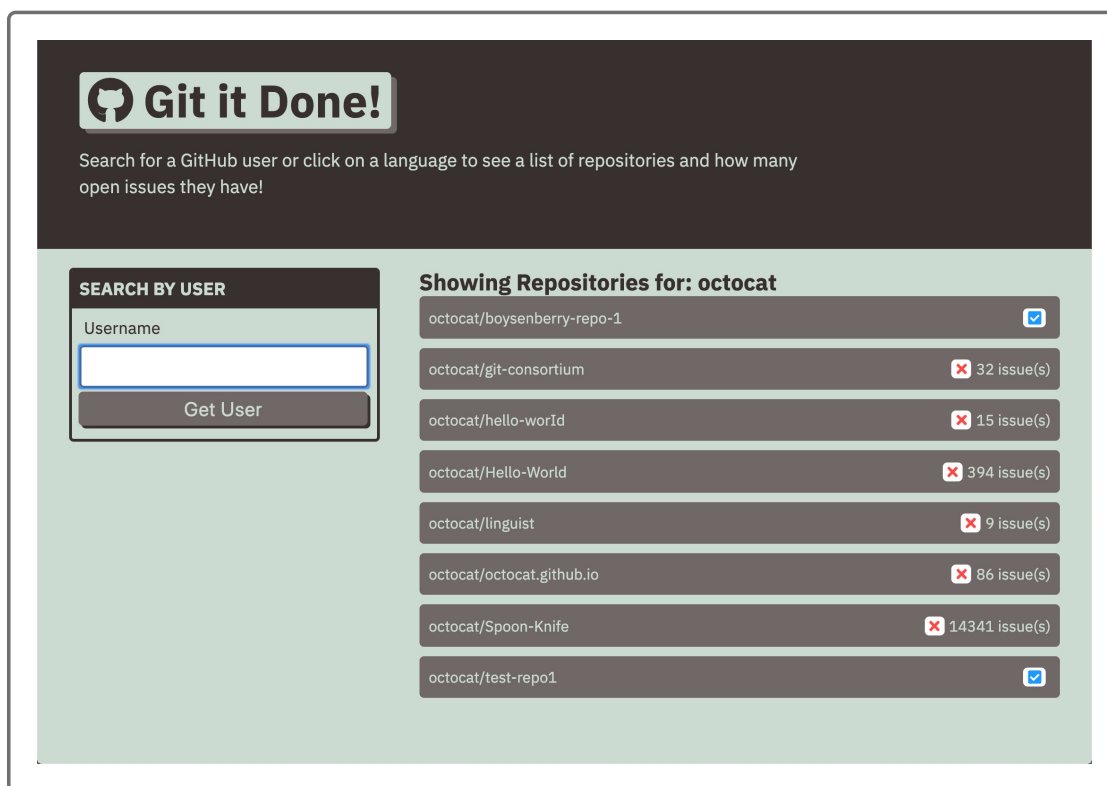
Now that we've created the function, let's set it up so that when the response data is converted to JSON, it will be sent from `getUserRepos()` to `displayRepos()`. Edit the `fetch()` callback code in the `getUserRepos()` function to look like this:

```
response.json().then(function(data) {  
  displayRepos(data, user);  
});
```

If we save `homepage.js`, refresh the page, and search for a GitHub user, we should see the result logged to the console from the `console.log()` statements in `displayRepos()`.

## Create an HTML Container for the Content

With the `displayRepos()` function now receiving data, we can update the function to start displaying data. But before that, try to answer one question. Where is this data getting displayed on the page? Take another look at the desired end result:



Okay, so it looks like the data will get displayed on the right column of the page. We already have the column set up, but we still need something unique in the HTML that we can select and write the data to. Let's update the HTML file so it has just that.

In `index.html`, update the right column to look like this:

```
<div class="col-12 col-md-8">
  <h2 class="subtitle">Showing Repositories for: <span id="repo-search">
  <div id="repos-container" class="list-group"></div>
</div>
```

Note that we've created two areas we'll write data to: We created a `<span>` element to write the search term to so that users know whose repositories they're looking at. We also created an empty `<div>` element that we'll write all of the repository data to.

But before we use JavaScript to write data into these two areas, let's create two more variables to reference these DOM elements at the top of `homepage.js`:

```
var repoContainerEl = document.querySelector("#repos-container");
var repoSearchTerm = document.querySelector("#repo-search-term");
```

Now we can start displaying the GitHub API response data on the page!

---

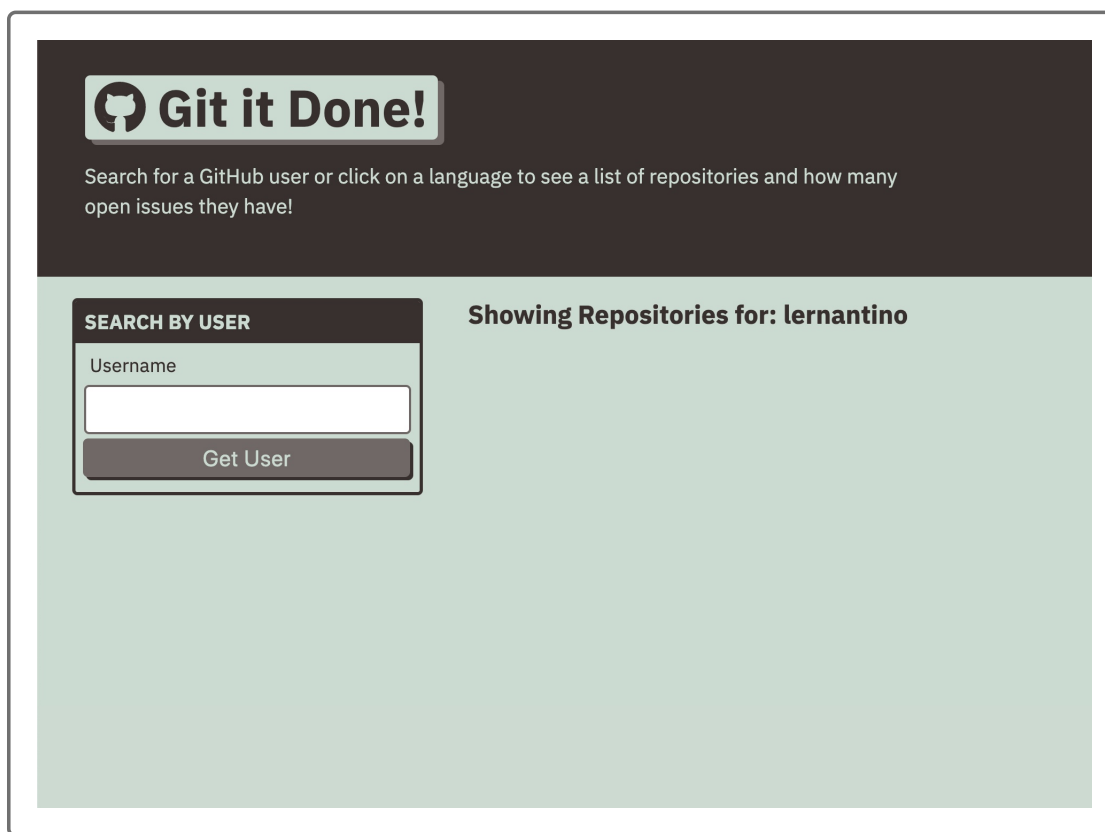
## Display Repository Data

When working with an app that displays data based on user input, we should always be sure to clear out the old content before displaying new content.

To do that, let's show the search term on the page by adding the following code to `displayRepos()`:

```
// clear old content
repoContainerEl.textContent = "";
repoSearchTerm.textContent = searchTerm;
```

As always, we should test the functions as we write them, not only to catch any bugs early on but also to affirm that we're doing something right! As a test, let's save `homepage.js` and try searching for a user. The page should display the name after a successful search response, like this image shows:



Now let's move on to the important part—displaying repository data to the page.

Add the following code to `displayRepos()`:

```
// loop over repos
for (var i = 0; i < repos.length; i++) {
  // format repo name
  var repoName = repos[i].owner.login + "/" + repos[i].name;

  // create a container for each repo
  var repoEl = document.createElement("div");
  repoEl.classList = "list-item flex-row justify-space-between align-c
```

```
// create a span element to hold repository name
var titleEl = document.createElement("span");
titleEl.textContent = repoName;

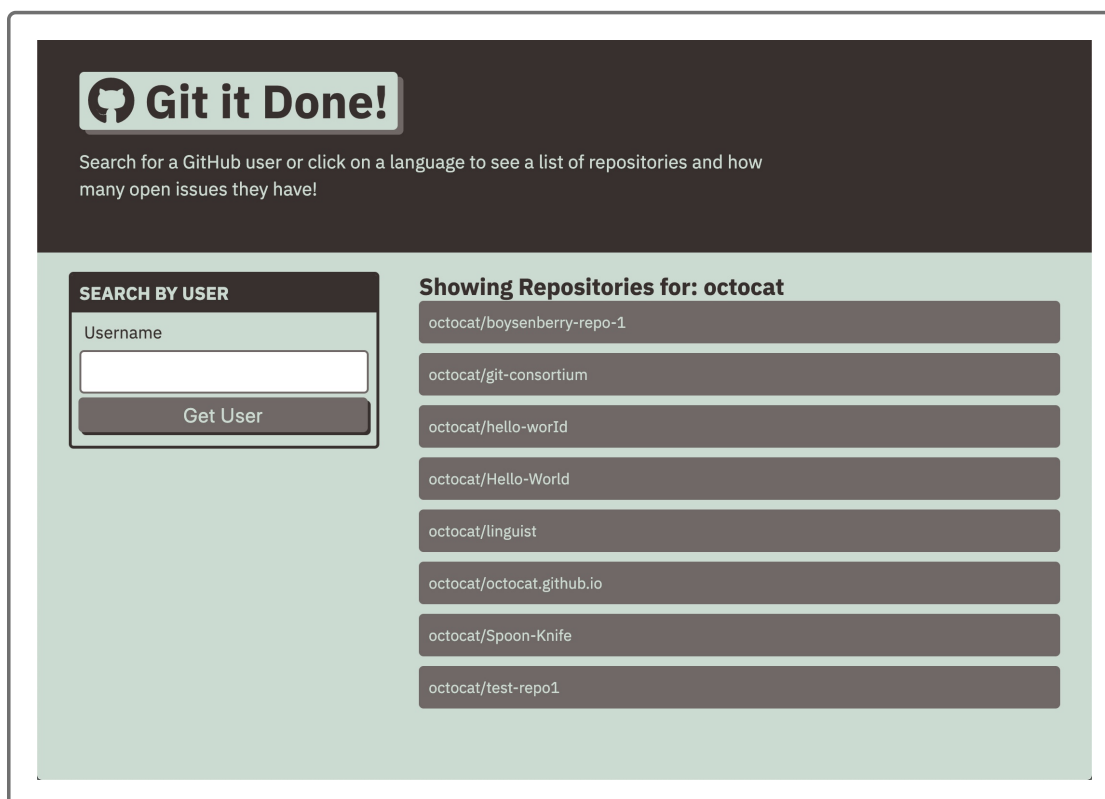
// append to container
repoEl.appendChild(titleEl);

// append container to the dom
repoContainerEl.appendChild(repoEl);
}
```

## IMPORTANT

Don't forget that the CSS classes come from Amiko's style sheet. Amiko has also included the font icon library called Font Awesome in the app!

Again, save `homepage.js` and try searching for a user. We should see something like this image:



How does this work? In the `for` loop, we're taking each repository (`repos[i]`) and writing some of its data to the page. First we format the appearance of the name and repository name. Next we create and style a `<div>` element. Then we create a `<span>` to hold the formatted repository name. We add that to the `<div>` and add the entire `<div>` to the container we created earlier.

We've done things like this in the past, except this time the data is coming from GitHub!

---

## Display Repo Issues on the Page

Now that we know the code works the way we want it to, we just need to write one more bit of information to the page: the number of issues for each repository.

Amiko wants to see not just the number of issues but a little icon next to the number of issues to help identify which repositories need help at the moment. We'll have to use an `if` statement for this, so let's update the code and implement it!

Update the `for` loop in `displayRepos()` to have this code right above the `repoContainerEl.appendChild()` method:

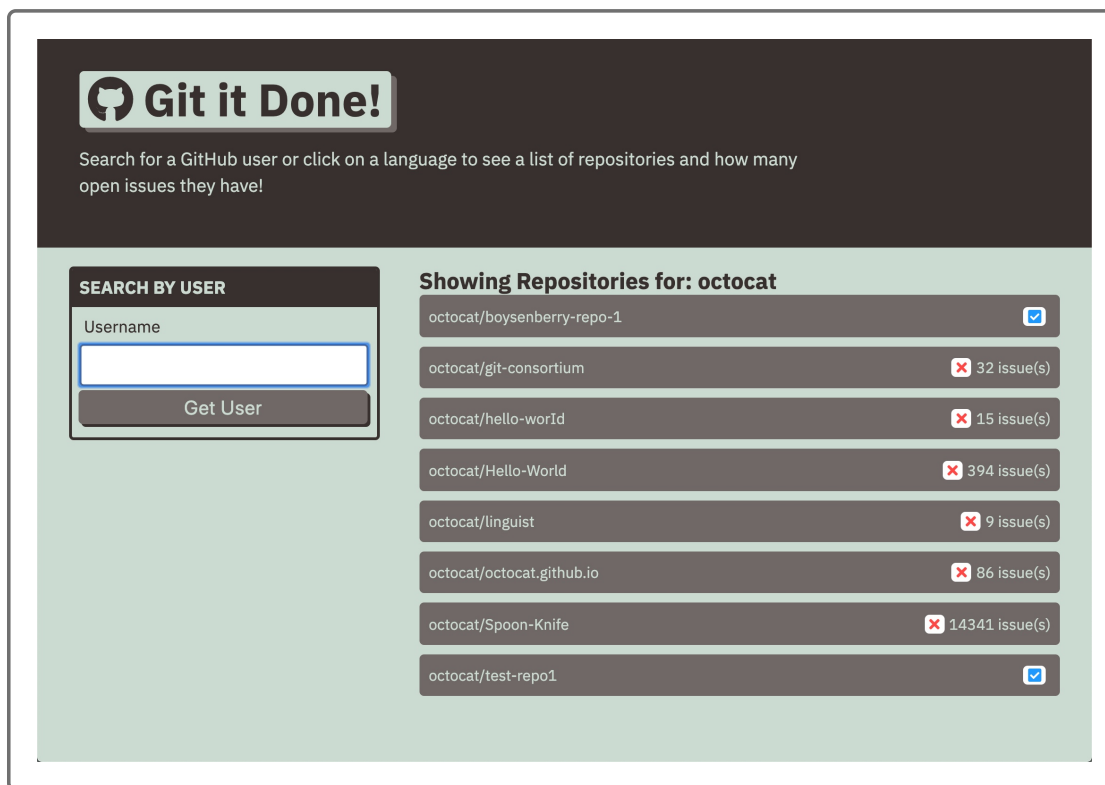
```
// create a status element
var statusEl = document.createElement("span");
statusEl.classList = "flex-row align-center";

// check if current repo has issues or not
if (repos[i].open_issues_count > 0) {
  statusEl.innerHTML =
    "<i class='fas fa-times status-icon icon-danger'></i>" + repos[i].
} else {
  statusEl.innerHTML = "<i class='fas fa-check-square status-icon icon"
}

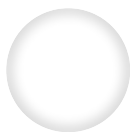
// append to container
repoEl.appendChild(statusEl);
```



Now if we try to search for a user, we'll get back not only the list of repositories but also the number of issues for each one, like this image shows:



We use an `if` statement to check how many issues the repository has. If the number is greater than zero, then we'll display the number of issues and add a red X icon next to it. If there are no issues, we'll display a blue check mark instead.



## REWIND

The font icons used here come from a service called Font Awesome. They offer a large selection of free font icons,

including icons for social networks, like the GitHub logo at the top of the page!

Great job on completing this type of functionality! The lessons you've learned from working on this app will be useful in all of your web development adventures—and now you can also add server-side APIs to your skill set.

Knowing how to work with server-side APIs opens up many possibilities for what you can build. Everything you've worked on until now has used data you created yourself. Now you can request data from other sources and use that instead. This doesn't come without potential hurdles, however, and we'll learn how to handle those next.

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