

# Getting Data from the Server

We need data to fill the items in our menu and sub-menu. For this purpose, we'll have to use JavaScript to connect to the server and fetch data.

## WE'LL COVER THE FOLLOWING ^

- How to Fetch Values?

In a real application, the values in the dropdown menu might not be hardcoded. For example, the image might change depending on which items are advertised more heavily on a given day (or even for the particular user).

## How to Fetch Values? #

For values that come from a server, we usually need JavaScript (note: that's not always the case. For example, you can fetch an image from an endpoint that serves different images depending on the request or other factors).

Since this course is about creating components, we won't be building or standing up an actual server. Instead, we'll build a fake server of sorts.

Let's start from the JavaScript line that gets the values and work our way back until it works end to end.

That line looks something like this:

```
helper.get(endpoint, data, callback)
```

What does this line mean?

- `helper`

- This is a library to facilitate making network requests. The native JavaScript way of doing so involves a lot more boilerplate, repetitive code, and is almost always replaced with an open-source solution.
- `get`
  - This specifies what method we'll be calling the endpoint with. Though there a number of HTTP methods, `get` and `post` are by far the most popular. So if you're just starting out, you can get away with just knowing these two. Broadly speaking, `get` is for getting information, `post` is for giving information.
- `endpoint`
  - This is the URL that our client will get the information from. Web servers need some way of communicating with the various browsers that the corresponding application might run on, so publicly available URLs are exposed to allow this.
- `data`
  - This value is like arguments to a method in programming. Servers might use (or might require) certain information to serve your request. For example, if the client is fetching a menu, we might want to specify which menu.
  - In a `get`, data is transferred by way of query parameters. What are query parameters? Let's say our endpoint is `server.com/menus`, and our data is `{menu: fashion}`. The request will be formatted to be `server.com/menu?menu=fashion`.
  - In a `post`, data is transferred to the body of a request. This is useful when we want to hide the data. For example, when you login on websites, they (hopefully!) use `post` requests since the body can be encrypted on the wire, but the URL cannot.
- `callback`
  - Finally, a callback is the function that's called by `helper` when the response is given. We can't just do `const value = helper.get(...)` because a network call is an asynchronous operation. This means it comes back in an uncertain amount of time.

Let's build this out, so it actually works. We'll be using this and building upon it in the rest of this lessons.

Let's come back to the dropdown menu in just a moment. For now as a test,

we just want to start with a blank page and have a server return “hello world” when we click anywhere.



```
const HOST = 'server.com/';

document.onclick = function() {
  api.get(HOST, {}, displayText);
}

function displayText(response) {
  document.body.innerHTML += response;
}
```

That’s all we need for client functionality, and it’s pretty straightforward: whenever the document (anywhere on the page) is clicked, we make the “api call” and our callback function appends some text to the HTML of the body. Now let’s create a fake server.

Output
JavaScript
HTML



**Note:** Try clicking around, and you’ll see that “hello world” is indeed appended. This has the bare minimum functionality for what we’ve set out to do (we haven’t even taken **data** into consideration), but it sets up the foundation for using a fake server so that we can focus on building our frontend components.

Now that we’ve introduced this concept, let’s get back to our dropdown menu.