**A Beginner Introduction to Using Axios for Requests -** [Emmanuel Unyime](https://medium.com/@ekimunyime?source=post_page-----e07fe9c87f54--------------------------------) Apr 22, 2021

**PREREQUISITES:**

1. Asynchronous JavaScript. Read my article [here](https://medium.com/swlh/a-quick-guide-to-writing-ajax-requests-for-json-files-a769b6be604e)
2. Promises in JavaScript
3. Objects In JavaScript
4. jQuery and Bootstrap might be used in this article.

Link to the Project Repository — [here](https://github.com/Uny1me/axios-project/).

**A BRIEF INTRODUCTION**

“In a nutshell, Axios is a JavaScript library used to make HTTP requests from [node.js](https://nodejs.org/en/) or XMLHttpRequests from the browser that also supports the [ES6 Promise API](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise). — [*What is Axios.js and why should I care?*](https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0)*By @Eric Kolleger.*To know more about Axios I’d recommend the article as a read.

What I want to do in this article is to teach you how to use Axios. While you can either use Axios on your Node.js applications or you can use it on a simple web app, in this article I will be using it on a simple web page.

**LET’S GET STARTED**

Start a project folder with an index.html file and index.js file and make sure to reference the script from the HTML file as well as the Axios link.

<script src=”https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

You can then go and paste this code in the body of the file:

<div class=”container pt-5">  
 <button onClick=”getRes()” class=”btn btn-danger”>GET</button>  
</div>

*(I am using bootstrap hence the use of classes: btn btn-danger and container)*

Starting this up we have on-screen a button with a function called getRes listening for its click event.

**THE GET REQUEST**

*In index.js*

// GET REQUEST  
function getRes() {  
 axios({  
 method: ‘get’,  
 url: ‘https://jsonplaceholder.typicode.com/todos'  
 })  
.then(result => console.log(result))  
.catch(error => console.error(error));  
}// Usually you output res i.e. response to the console but to get data Object we use res.data

**Explanation:**

In the function getRes, we call Axios with the syntax *axios({})*. Between the curly braces, we can choose the method of data communication we want to use e and then specify the URL (location) from which the data communication happens.

This is followed by the **“.then()”** statement, between the parentheses, is code that is run strictly if the request was successful, this often involves referencing the parts of the data that has been received. Data received is usually like a JavaScript object and can be saved to any name, above we choose ‘result’

Next is the **“.catch()”**statement, between its parentheses, is code executed if the request failed, this error can be printed to the console after being saved into a variable which in this case is “error”.

Since the link above is valid, the requests for the to-do lists link stored in the link is successful, this means that the console should have an output like:

Text

Description automatically generated

However getting this as a response doesn’t do much for us yet, we have to extract the data we need. So going back to the code we can use our knowledge of objects to log to the console the data stored within the response object:

.then(result => console.log(result.data))

and then we can get an array of 200 values.

Like shown:

Text

Description automatically generated

You are not restricted the only logging output or inputs into the console, you can choose to manipulate the DOM as you see fit. Go ahead and replace your then statement with this and see what happens:

**.then(res => alert(‘you successfully tested a get request with Axios’))**

I don’t know bout you but 200 seems like a long number to shorten it so we can limit the data amount being received with a few lines of code. Within the braces of the Axios request, we go ahead and limit the data like this:

params: {  
 \_limit: 5  
}

So our total function looks like this:

axios({  
 method: ‘get’,  
 url: ‘https://jsonplaceholder.typicode.com/todos',  
 params: {  
 \_limit: 5  
 }  
})  
.then(res => console.log(res))  
.catch(err => console.error(err));

However, we don’t always have to write this fully, we can shorten it out. We can do this like this:

axios.get(‘https://jsonplaceholder.typicode.com/todos?\_limit=5')  
.then(res => console.log(res))  
.catch(err => console.log(err))

The limit is written in the URL i.e. ‘?\_limit=5’

**OUTPUTTING DATA TO THE DOM**

Text

Description automatically generated

To understand how to output data to the DOM we have to look at the structure of what we have received, i.e. the result we asked the browser to log into the console using our then statement, this is where an understanding of Objects will help.

Looking at the image we can see that it has a status of 200 which means it was a successful request, let us say we wanted to see just 200 in the console. The parent object of all these nested items is what we have called ‘res’, so if we simply write in our then statement:

.then((res) => {  
 console.log(res.status);  
 })

The structure looks a bit different compared to our other then statement, but this way we can write multiple executable functions if Axios was successful.

Run the request and in your console you should see 200, so if we wanted to see 200 in the DOM we can simply append res.status to an HTMLelement. For example:

document.getElementById(‘root’).innerHTML = `${res.status}`  
//Or using jQuery shorthand:  
$(‘#root’).text(`${res.status}`)

**A complex example:**

We can likewise write this in then statement

**$(‘#content’).html(`  
<div class=”card col-7 p-0 mt-4">  
<h5 class=”card-header”>Received from user ${res.data[0].userId}</h5>  
<div class=”card-body”>  
<p class=”lead” id=”inner-text”>The todo is ${res.data[0].title}</p>  
</div>  
<div class=”card-footer lead”>Completed: <b>${res.data[0].completed}</b></div>  
</div>  
`)**

**Explanation:**This is the beauty of JavaScript, being able to create HTML elements given certain conditions. Here we simply state that if the Axios request is successful, the HTML element should be created and put into the element with the content, i.e. line 1 of code above (The vanilla equivalent of the jQuery shorthand above is to use innerHTML. i.e.

document.getElementById(‘content’).innerHTML = ‘’

Try it and notice what happens. Recall however that we are asking for 5 items through our request, so what if we wanted to see all 5?

**An even more complex example:**

Text

Description automatically generated

**Explanation:**

The code is littered with comments that make it self explanatory to a point, We simply use a for loop to create elements and for each data in each instance. We then assign data (to created elements) gotten from axios requests.

**The POST request, as well as Patch, will be covered in part II of this article.**