**What is Axios.js and why should I care? -** [Eric Kollegger](https://medium.com/@MinimalGhost?source=post_page-----7eb72b111dc0--------------------------------) May 14, 2018

As a largely self-taught developer, I’ve put the wide world of free/paid internet tutorials through it’s paces. Something I see pop up again and again in tutorials performing HTTP requests is the use of a JavaScript library called [Axios](https://github.com/axios/axios). I wanted to understand why such a wide range of web development educators were making use of this seemingly inconspicuous library. So I decided to dig in to deliver a very simple, straightforward explanation of what Axios offers and why it’s (probably) worth using.

**What it does**

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). Great, so from that we gather it does something that we can already do and that has recently been made significantly better… So why bother?

**What it aims to improve: .fetch()**

Fetch uses a two-step process when dealing with JSON data; after making an initial request you’ll then need to call the .json() method on the response in order to receive the actual data object.

If I were to have a fetch request to get back an object of relevant drivers for my carpool sourcing app, I might have something like the following:

const baseUrl = '<http://localhost:3000/api/v1'>fetch(`${baseUrl}/drivers`).then(res => res)

This would return a pending promise and response object letting me know my request was successful, but where’s my data?! We have to remember to pass our response to the .json() method as well.

const baseUrl = '<http://localhost:3000/api/v1'>fetch(`${baseUrl}/drivers`).then(res => res.json())

Now that’s not a big deal, like many things it really comes down to syntax memorization. But as developers, we’re simultaneously the laziest and most efficient creatures on the planet — so any kind of shorthand or time saver is worth consideration, enter Axios.

**Better error handling**

There’s some extra work we need to put in on our end (again, it could be argued that this is just a matter of syntax memorization) for the desired result when it comes to properly logging response errors with .fetch(). [MDN](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#Checking_that_the_fetch_was_successful) explains:

*A*[*fetch()*](https://developer.mozilla.org/en-US/docs/Web/API/GlobalFetch/fetch)*promise will reject with a [TypeError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypeError" \t "_blank) when a network error is encountered or CORS is misconfigured on the server side, although this usually means permission issues or similar — a 404 does not constitute a network error, for example.*

Let’s assume there’s something wrong with our baseUrl variable. The expectation of the output for the following code block would be *error*, but instead it’s *ok.*Why is this?

const baseUrl = '<http://localhost:3000/api/v1'>fetch(`${baseUrl}/drivers`)  
 .catch(error => console.log('ok'))  
 .then(res => console.log('error'));

It turns out fetch() only rejects promises in relatively rare networking error situations like a DNS lookup failure. Luckily fetch() provides a simple ok flag that indicates whether or not the status code of a response is in the successful range. Generally what happens is you’ll end up writing a one-size-fits-all error handling function to be used on all of your fetch() calls.

**Installing Axios**

Grabbing new libraries is easy as pie these days (Mmm.. rhubarb), I prefer npm. In the root of your project folder open terminal, run npm install axios and you’re all set. Now to achieve the same fetch request we performed earlier, we can invoke axios.get() in place of our fetch method like so:

const baseUrl = '<http://localhost:3000/api/v1'>axios.get(`${baseUrl}/drivers`).then(res => res)

Presto! By using axios we remove the need to pass the results of the HTTP request to the .json() method. Axios simply returns the data object you expect straight away. Additionally, any kind of error with an HTTP request will successfully execute the .catch() block right out of the box.

Here’s a brief rundown of all it’s primary features, including a few I didn’t cover direct from the GitHub page:

* Make [XMLHttpRequests](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) from the browser
* Make [http](http://nodejs.org/api/http.html) requests from node.js
* Supports the [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) API
* Intercept request and response
* Transform request and response data
* Cancel requests
* Automatic transforms for JSON data
* Client side support for protecting against [XSRF](http://en.wikipedia.org/wiki/Cross-site_request_forgery)

In summation, Axios is a quality of life improvement more than anything else. But making many small, incremental quality of life adjustments to ones workflow can drastically improve development quality and speed.