To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

Jason Arnold Follow

Feb 9, 2017 · 4 min read · Listen

Fetch vs. Axios.js for making http requests

That is a handsome face!

One of the final sections of Stephen Grider’s excellent ‘ES6 Javascript: The Complete Developer’s Guide’ course on Udemy.com discusses the .fetch() method and some if its shortcomings. He points out that there are some things with .fetch() that aren’t ideal and suggests that there are other options out there for making HTTP requests. One of those options is axios.js. I had not heard of Axios before so this seemed like a great opportunity to do a little digging and see what I could come up with. (Since this is referencing material from Stephen’s course, I am using examples and conventions similar to his.)

Axios is a Javascript library used to make http requests from node.js or

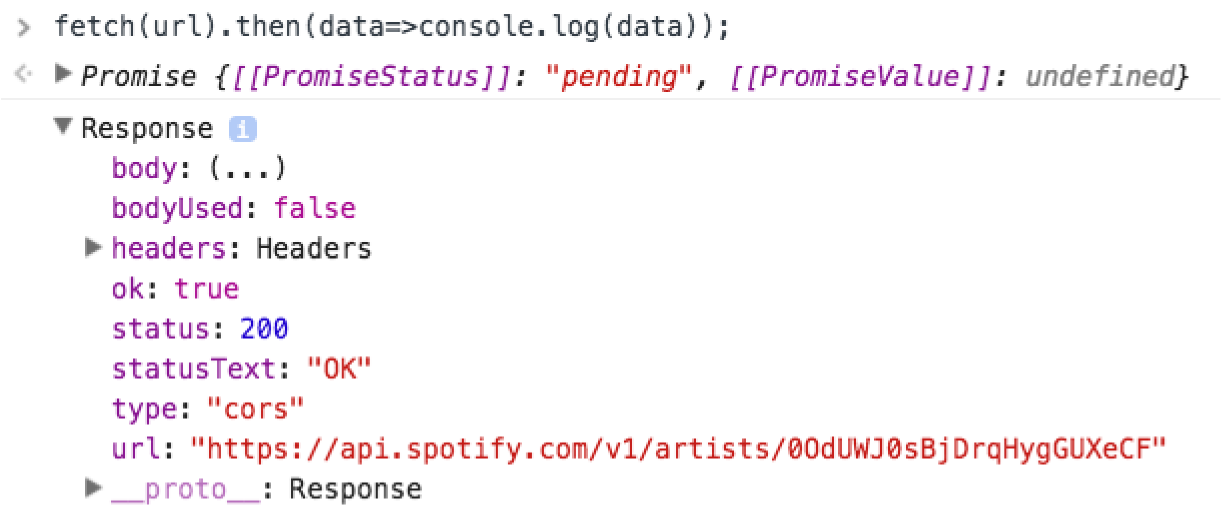
To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

If you use .fetch() there N data. The first is to make the actual request and then the second is to call the .json() method on the response.

Update 06/27/17: As of May 29th, Spotify now requires authentication for all requests to their API, so the below examples will not work as written.

Here is a simple example using the Spotify API. I set the url as a variable and then pass it to fetch and set the .then() callback to console the data that gets returned from the request.

```
const url =
'https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF'

fetch(url).then(data=>console.log(data));
```



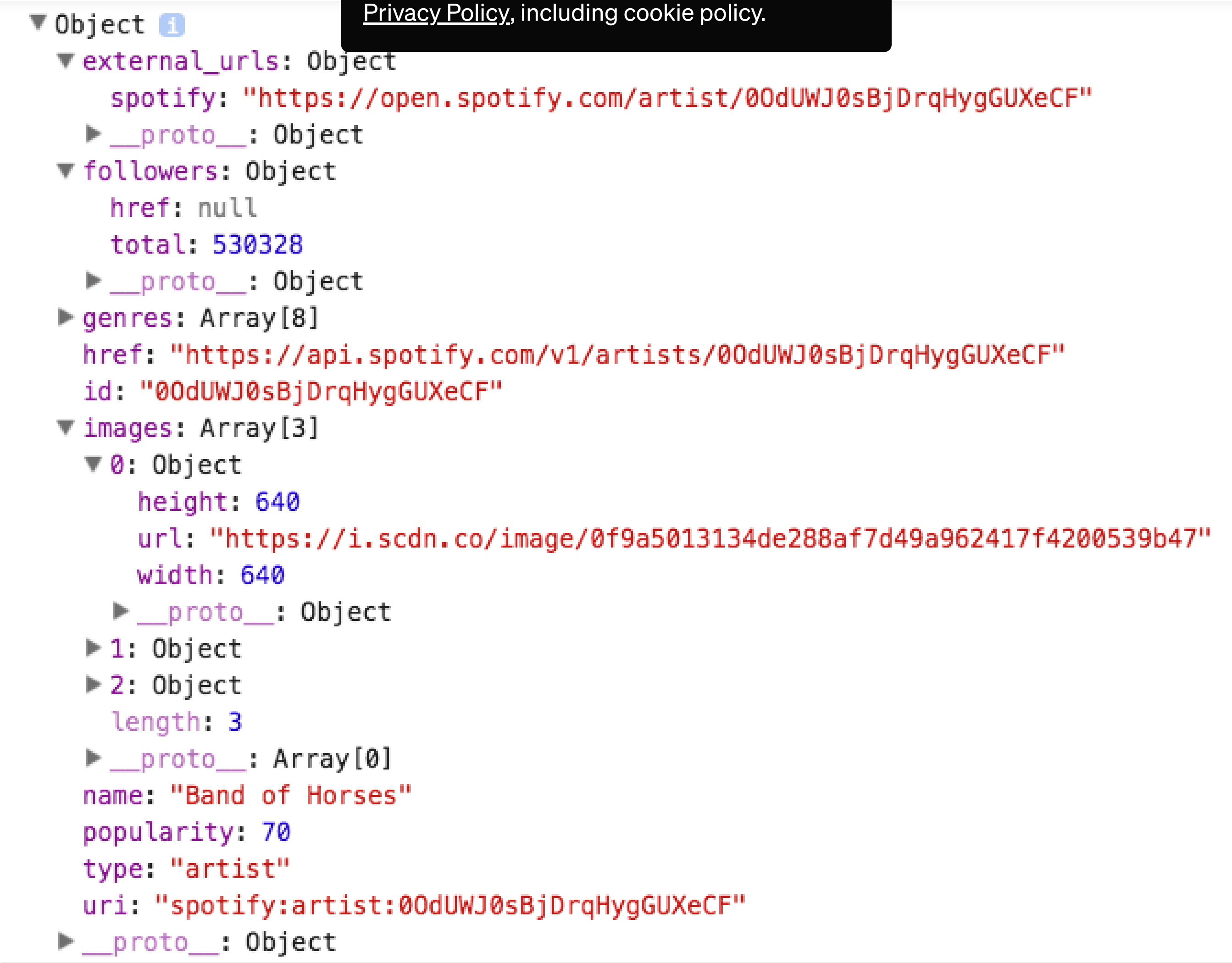
```
> fetch(url).then(data=>console.log(data));
<· ▶Promise {[[PromiseStatus]]: "pending", [[PromiseValue]]: undefined}
  ▼Response ℹ
     body: (...)
     bodyUsed: false
    ▶headers: Headers
     ok: true
     status: 200
     statusText: "OK"
     type: "cors"
     url: "https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF"
    ▶__proto__: Response
```

The default response from the .fetch() method.

That’s great, but this is not the data you are looking for. That is the response from the server letting you know that your request went through just fine. Great, but you can’t do much with that.

To get to that data, you have to pass it to .json() first and then you can see it.

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.



```
▼ Object
  ▼ external_urls: Object
      spotify: "https://open.spotify.com/artist/0OdUWJ0sBjDrqHygGUXeCF"
    ▶ __proto__: Object
  ▼ followers: Object
      href: null
      total: 530328
    ▶ __proto__: Object
  ▶ genres: Array[8]
    href: "https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF"
    id: "0OdUWJ0sBjDrqHygGUXeCF"
  ▼ images: Array[3]
    ▼ 0: Object
        height: 640
        url: "https://i.scdn.co/image/0f9a5013134de288af7d49a962417f4200539b47"
        width: 640
      ▶ __proto__: Object
    ▶ 1: Object
    ▶ 2: Object
      length: 3
    ▶ __proto__: Array[0]
    name: "Band of Horses"
    popularity: 70
    type: "artist"
    uri: "spotify:artist:0OdUWJ0sBjDrqHygGUXeCF"
  ▶ __proto__: Object
```

The result after passing the response to the .json() method.

Now, this is the data from Spotify that we wanted.

Let’s see how this is handled with Axios.

The first step in using Axios is installing Axios. You can use npm if you want to run axios in node or a cdn if you want to run it in your browser.

```
npm install axios
```

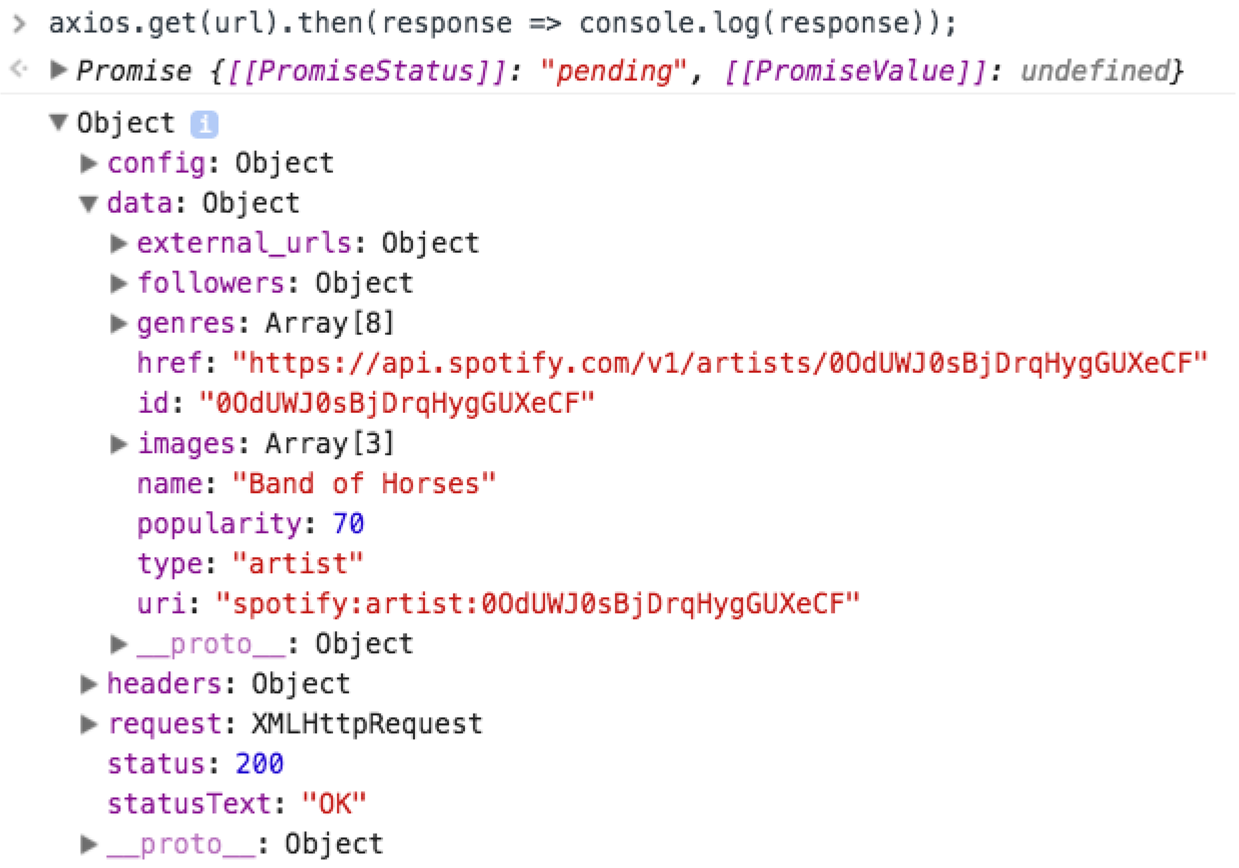
OR

```
https://cdnjs.cloudflare.com/ajax/libs/axios/0.15.2/axios.min.js
```

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

Then, in my console, I as o the `axios.get()` method.

```
const url =
'https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF'

axios.get(url).then(response => console.log(response));
```



```
> axios.get(url).then(response => console.log(response));
<· ▶ Promise {[[PromiseStatus]]: "pending", [[PromiseValue]]: undefined}
  ▼ Object i
    ▶ config: Object
    ▼ data: Object
      ▶ external_urls: Object
      ▶ followers: Object
      ▶ genres: Array[8]
        href: "https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF"
        id: "0OdUWJ0sBjDrqHygGUXeCF"
      ▶ images: Array[3]
        name: "Band of Horses"
        popularity: 70
        type: "artist"
        uri: "spotify:artist:0OdUWJ0sBjDrqHygGUXeCF"
      ▶ __proto__: Object
    ▶ headers: Object
    ▶ request: XMLHttpRequest
      status: 200
      statusText: "OK"
    ▶ __proto__: Object
```

So by using axios you can cut out the middle step of passing the results of the http request to the .json() method. Axios just returns the data object you would expect.

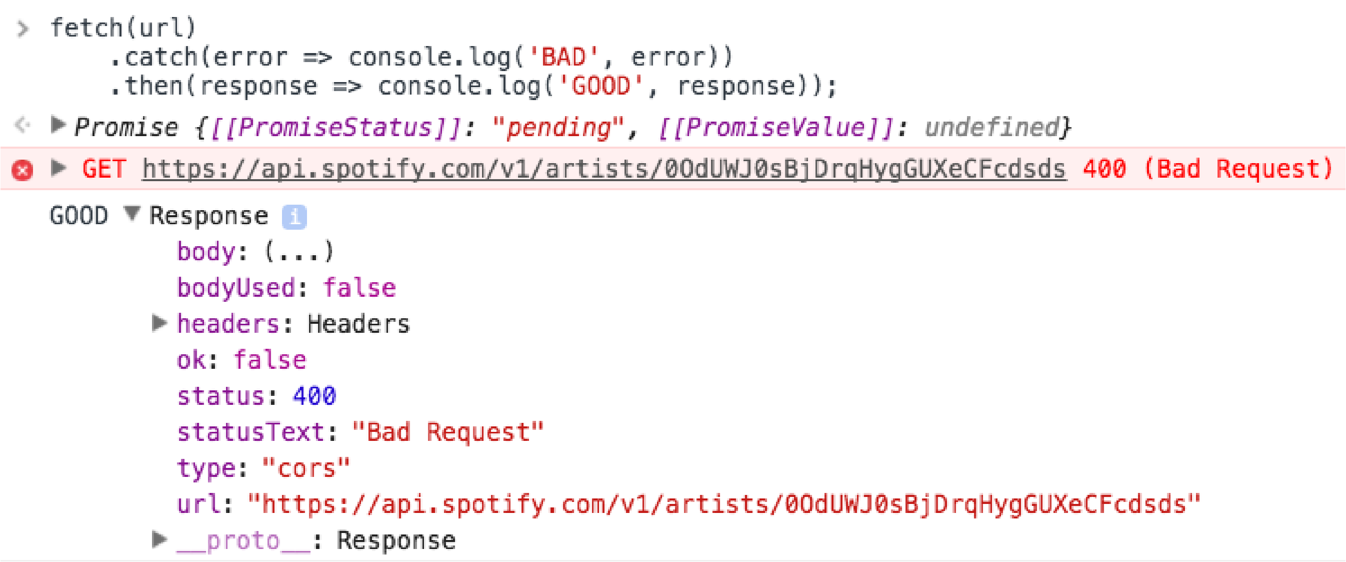
The second issue that Stephen brings up is how .fetch() handles error responses. Logically you would think that if .fetch() gets an error it would enter the .catch() block and return anything there, right? Not necessarily. Here is an example.

I have altered my `url` variable from the previous examples so that it is now incorrect. I would expect a 400 error at this point and for my .fetch() to go into the .catch() block

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

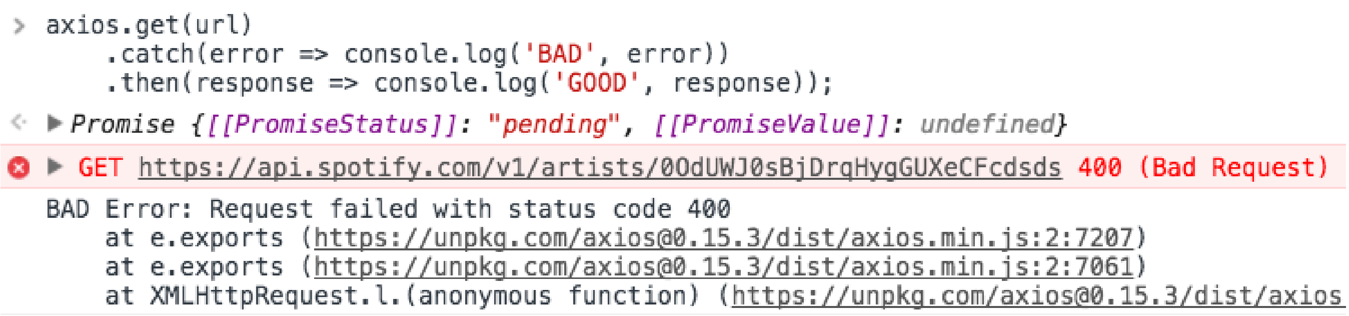
```
const url =
'https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCFcdsds';

fetch(url).catch(error => console.log('BAD', error)).then(response
=> console.log('GOOD', response));
```

I've added the 'BAD' and 'GOOD' strings in the responses to clarify what is happening here.



```
> fetch(url)
      .catch(error => console.log('BAD', error))
      .then(response => console.log('GOOD', response));
< ▶ Promise {[[PromiseStatus]]: "pending", [[PromiseValue]]: undefined}
⊗ ▶ GET https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCFcdsds 400 (Bad Request)
  GOOD ▼ Response ℹ
           body: (...)
           bodyUsed: false
         ▶ headers: Headers
           ok: false
           status: 400
           statusText: "Bad Request"
           type: "cors"
           url: "https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCFcdsds"
         ▶ __proto__: Response
```

I get the 400 response code but, as you can see by the 'GOOD' string in the console, the .then() block was executed. How does Axios handle this? The way you would probably expect. You get any kind of error with the http request and the .catch() block is executed.



```
> axios.get(url)
      .catch(error => console.log('BAD', error))
      .then(response => console.log('GOOD', response));
< ▶ Promise {[[PromiseStatus]]: "pending", [[PromiseValue]]: undefined}
⊗ ▶ GET https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCFcdsds 400 (Bad Request)
  BAD Error: Request failed with status code 400
      at e.exports (https://unpkg.com/axios@0.15.3/dist/axios.min.js:2:7207)
      at e.exports (https://unpkg.com/axios@0.15.3/dist/axios.min.js:2:7061)
      at XMLHttpRequest.l.(anonymous function) (https://unpkg.com/axios@0.15.3/dist/axios
```

The 'BAD' string is there and the error is logged to the console.

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.