**4 Simple Steps to Get Started with the Fetch API -** [Kyle DeGuzman](https://medium.com/@kyledeguzmanx?source=post_page-----2f3aafaca17d--------------------------------) Jan 5, 2022

Today we are learning how to fetch resources using the Fetch API. Using this API, our own applications can communicate with third-party applications to retrieve meaningful resources and functionality that we can use to enhance our own products. All we need to get started is a URL or API that will send us back some data.

For example:

* using the Stripe API or PayPal API we can process transactions
* using the Unsplash API we can retrieve images
* using many different independent APIs we can retrieve all sorts of meaningful information

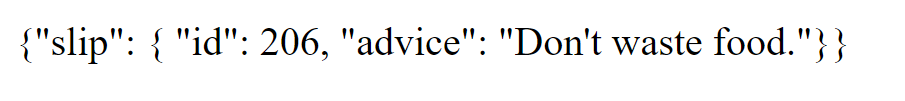
Let’s get started.

**Find An API**

Our first step is to find an API. We need a URL, or path, that will return data when queried. Here are a few examples:

**[1]**[**Advice API**](https://api.adviceslip.com/advice)

This API generated advice.



This is what the stream of data looks like. As you can see, it’s a JavaScript object. The object has the property slip. And slip is also an object. slip has the properties: id and advice.

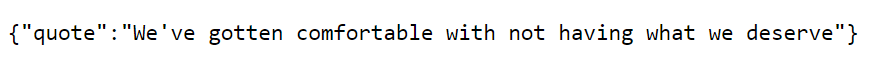
We want the contents of the advice property. We would simply access it using dot-notation or bracket-notation. If you don’t know how to do that, check out this straightforward [tutorial](https://www.w3schools.com/js/js_object_properties.asp)by W3Schools.

To access it, it would look something like: data.slip.advice or data["slip"]["advice"]

If you click the link to access the API, refresh it. Notice how the API provides new data upon every refresh.

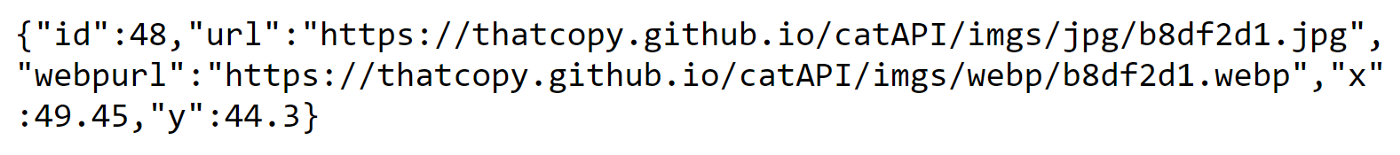
**[2] [kanye.rest API](https://api.kanye.rest/" \t "_blank)**

This API provides quotes.



This is a more simple example. Again, this is an object. We must use dot-notation or bracket-notation to access the quote. It would look like data.quote.

**[3]**[**Cat Photos API**](https://thatcopy.pw/catapi/rest/)



Here is another example. It would look something like data.url

Also, understand that the information returned will not always look like this. Sometimes it will return an array of objects. Sometimes it will return objects within objects. Sometimes it will just return text. It’s important you understand JavaScript data structures if you want to properly retrieve the information you want.

**Fetch Resources**

When you have the URL of your API, you can do the following:

fetch("<https://api.kanye.rest/>")

or

const URL = "<https://api.kanye.rest/>"  
fetch(URL)

It’s the same idea, just different appearances.

With this code, you are making an HTTP request and communicating with a web server. You are retrieving information from an external source. When this happens, JavaScript doesn’t know how long this fetching process will take so it performs this task asynchronously.

This means JavaScript can continue with the following lines of code, instead of pausing and waiting for the fetch process to finish. There’s no block/pause. And once this fetch process does conclude and we have our results, JavaScript will come back to it and finish the job. This is the entire idea behind *JavaScript Promises*.

Coming back to the code, the response of a fetch is a JavaScript Promise.

console.log(fetch(URL));

The code above will return a promise. And if you understand promises, you know that promises need to be resolved. Resolved is basically saying “once the promise has been fulfilled or the task has been finished, what do we do next?” So once the fetch is complete, what do we do next?

**Resolve**

We will use .then() to tell JavaScript what to do next.

fetch("<https://api.adviceslip.com/advice>")  
 .then(data =>{  
 return data.json();  
 })

or you might feel more comfortable if it did not use arrow functions. Alternatively, it can look like this:

fetch("<https://api.adviceslip.com/advice>")  
 .then(function(data){  
 return data.json();  
 })

As you can see, .then() contains a function. And that function has a parameter data. (*Note: data was arbitrarily named. You can name it whatever you want.*)data is the value returned from fetch().

data is the HTTP response. So when we write data.json() we are parsing the HTTP response as JSON.

Other options include:

* data.text()
* data.blob()
* data.formData()
* and more. Check out the Methods section in the MDN Web Docs for the [Response Interface](https://developer.mozilla.org/en-US/docs/Web/API/Response)

In this example, we’re using json. And if you don’t understand why, do this:

fetch("<https://api.adviceslip.com/advice>")  
 .then(data =>{  
 console.log("data: " + data);  
 console.log("json: " + data.json());  
 return data.json();  
 })

Compare them.

**Resolve Again**

If you attempted the sample code above, you would see that return data.json returns a promise. And what did we say earlier?

if you understand promises, you know that promises need to be resolved.

So, we have to resolve it. What do we do next if the HTTP response was successfully parsed to JSON? We can finally access the content of the JSON file!

fetch("<https://api.adviceslip.com/advice>")  
 .then(data =>{  
 console.log("data: " + data);  
 console.log("json: " + data.json());  
 return data.json();  
 })  
 .then(data => {  
 quote.innerHTML = '"' + data.slip.advice + '."';  
   
 })

Again, our .then() contains a function with the parameter called data which was arbitrarily named. To access the property that we want, we do data.slip.advice. That process was outlined earlier.

In this example, we are manipulating the DOM so we can display the advice on our web page. And every time the web page is refreshed, a new quote of advice will be displayed.

If you want to see how that works, there are CodePen samples available at the bottom. There are two sites for quote generators. And the cat site is a cat photo generator. Rather than manipulating the innerHTML, we are changing the src attribute’s value of an img element. Check that out below.

**Extra**

Before we finish off, I want to show you one last thing: another format for the Fetch API.

If you have understood the logic behind everything we talked about, this should be straightforward.

async function getData(){ const fetchResponse = await fetch(apiURL); const data = await fetchResponse.json(); console.log(data.slip.advice);}

This is another way you might see Fetch being used by other developers.

Notice two things: async and await. async is basically there to state that await will be used somewhere in the function. await is there to tell JavaScript, in this specific case, that it should wait for the fetch() to finish before resuming to the next line of code.

Why? Well, look at the next line of code. The next line of code completely depends on the current line of code. The next line of code does fetchResponse.json() which parses fetchResponse into a JSON format. fetchResponse can only exist if the current line of code finishes. Right?

As you can see await is used again, in line 3. This is because line 4 completely depends on line 3 finishing. With line 3, data is in a JSON format that makes the data usable. And we access that data in line 4, using dot-notation. It’s the same idea as the other format.

For more information on how you can use the Fetch API, check out the [Google Developer Blog](https://developers.google.com/web/updates/2015/03/introduction-to-fetch) and the [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API).

If you want to check out some sample code to see the Fetch API in action, run the CodePens below. And finally, if you want to experiment by yourself, there is a compilation of [public APIs](https://github.com/public-apis/public-apis) you can use available on GitHub.

**Kanye West Quote API**

**Advice API**

**Cat Photo API**

*More content at*[*plainenglish.io*](http://plainenglish.io/)*. Sign up for our*[*free weekly newsletter*](http://newsletter.plainenglish.io/)*. Get exclusive access to writing opportunities and advice in our*[*community Discord*](https://discord.gg/GtDtUAvyhW)*.*