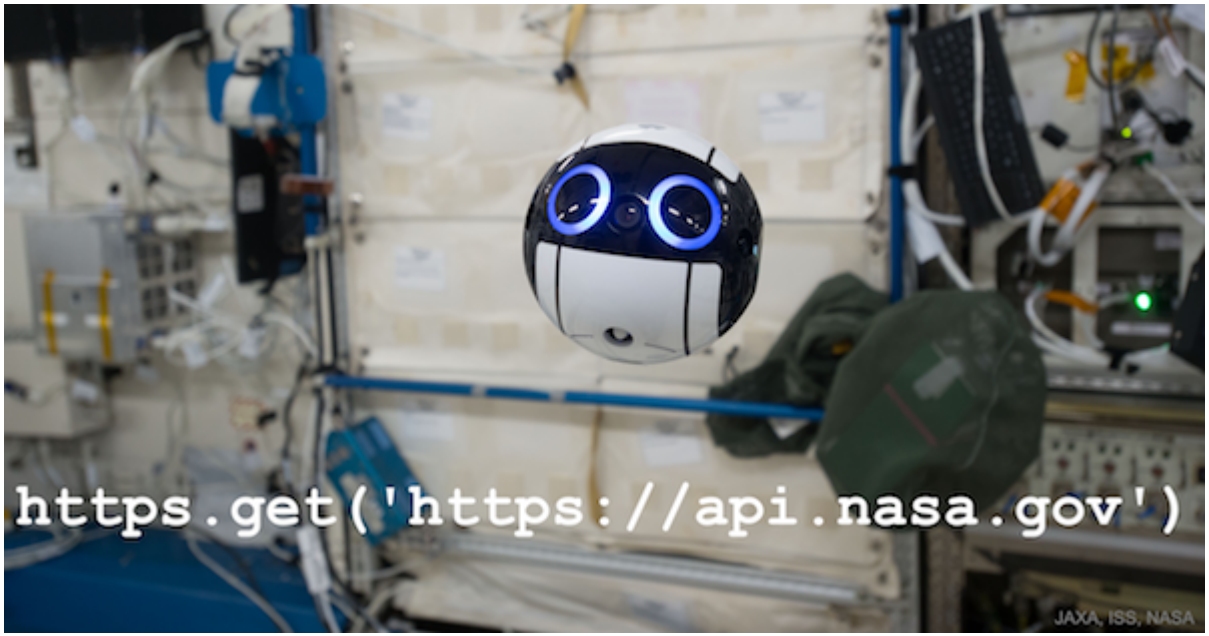
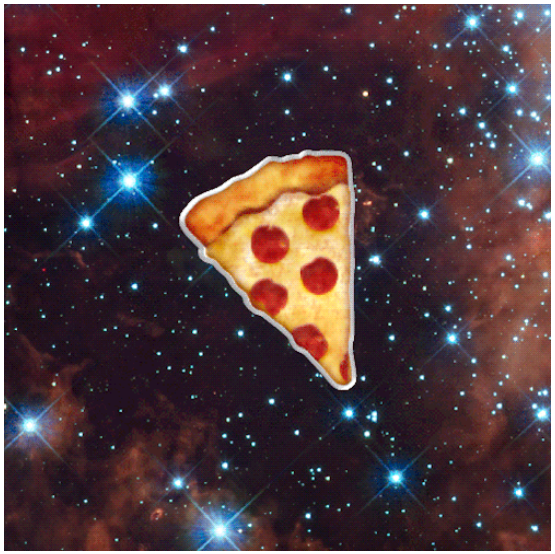
BY SAM AGNEW • 2017-08-09

5 Ways to Make HTTP Requests in Node.js



Making HTTP requests is a core functionality for modern languages and one of the first things many developers learn when acclimating to new environments. When it comes to Node.js there are a fair amount of solutions to this problem both built into the language and by the community. Let's take a look at some of the most popular ones.

We'll be using NASA's Astronomy Picture of the Day API as the JSON API that we are interacting with in all of these examples because space is the coolest thing ever.



Before moving on, make sure you have up to date versions of Node.js and npm installed on your machine.

HTTP – the Standard Library

First on our hit parade is the default `HTTP` module in the standard library. With this module, you can just plug and go without having to install external dependencies. The downside is that it isn't very user friendly compared to other solutions.

The following code will send a `GET` request to NASA's API and print out the URL for the astronomy picture of the day as well as an explanation:

```
1  const https = require('https');
2
3  https.get('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY', (resp) => {
4    let data = '';
5
```

```
  // A chunk of data has been recieved.
  resp.on('data', (chunk) => {
    data += chunk;
  });

  // The whole response has been received. Print out the result.
  resp.on('end', () => {
    console.log(JSON.parse(data).explanation);
  });

}).on("error", (err) => {
  console.log("Error: " + err.message);
});
```

Much of the `HTTP`, and the `HTTPS`, module's functionality is fairly low-level. You're required to receive response data in chunks rather than just providing a callback function to be executed as soon as all of the data is received. You also need to parse the response data manually. This is fairly trivial if it is JSON formatted, but it is still an extra step.

One other problem is that this module does not support `HTTPS` by default, so we need to require the `https` module instead if the API we are using communicates over `HTTPS`.
It may take a bit more effort to get the data you want, but is a great utility if you don't want to add too many dependencies to your codebase or want access to its low level functionality.

Request

Request is a simplified HTTP client comparable to Python's requests library. This library is much more user friendly than the default `http` module and has been considered a go-to for the community for several years.

This has been my personal choice since I've started using Node.js, and is great for quickly getting things done. Unlike the `http` module, you will have to install this one as a dependency from npm.

Run the following in your terminal from the directory you want your code to live in:

```
npm install request@2.81.0
```

You'll see that you need much less code to accomplish the same task that we did above:

```
const request = require('request');

request('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY', { json: true }, (err, res, body) => {
  if (err) { return console.log(err); }
  console.log(body.url);
  console.log(body.explanation);
});
```

Request is a fantastic option if you just want an easy to use library that deals with HTTP requests in a sane way. If you want to use Promises, you can check out the request-promise library.

Axios

Axios is a Promise based HTTP client for the browser as well as node.js. Using Promises is a great advantage when dealing with code that requires a more complicated chain of events. Writing asynchronous code can get confusing, and Promises are one of several solutions to this problem. They are even useful in other languages such as Swift.

To install Axios from npm, enter the following command in your terminal:

```
npm install axios@0.16.2
```

The following code will accomplish the same task of logging the URL to and of explaining the astronomy picture of the day:

```
const axios = require('axios');

axios.get('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY')
  .then(response => {
    console.log(response.data.url);
    console.log(response.data.explanation);
  })
  .catch(error => {
    console.log(error);
  });
```

Axios even parses JSON responses by default. Pretty convenient! You can also see that error handling is done with `.catch()` since we are using promises now.

You can even make multiple concurrent requests with `axios.all` if you wanted to do something like get the astronomy picture of two different days at once:

```
var axios = require('axios');

axios.all([
  axios.get('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY&date=2017-08-03'),
  axios.get('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY&date=2017-08-02')
]).then(axios.spread((response1, response2) => {
  console.log(response1.data.url);
  console.log(response2.data.url);
})).catch(error => {
  console.log(error);
```

```
11 });
```

Asynchronous code can easily become over complicated and unpleasant to deal with, and the way Axios tackles this problem may make your life easier in the long run.

SuperAgent

Similarly to Axios, SuperAgent is another popular library primarily used for making AJAX requests in the browser but works in Node.js as well. Install SuperAgent with the following command:

```
1 npm install superagent@3.5.2
```

What is cool about SuperAgent is that you have other useful functions that you can chain onto requests such as `query()` to add parameters to the request. We've been just manually adding them in the URL in the previous examples, but notice how SuperAgent gives you a function to do this:

```
1 const superagent = require('superagent');
2
3 superagent.get('https://api.nasa.gov/planetary/apod')
4 .query({ api_key: 'DEMO_KEY', date: '2017-08-02' })
5 .end((err, res) => {
6   if (err) { return console.log(err); }
7   console.log(res.body.url);
8   console.log(res.body.explanation);
9 });
```

Just like with Axios you don't have to parse the JSON response yourself, which is pretty cool.

Got

Got is another choice if you want a more lightweight library. It is also available to use in Twilio Functions.

Again, install Got with npm:

```
npm install got@7.1.0
```

Similarly to Axios, Got works with Promises as well. The following code will work as the rest of the examples do:

```
const got = require('got');

got('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY', { json: true }).then(response => {
  console.log(response.body.url);
  console.log(response.body.explanation);
}).catch(error => {
  console.log(error.response.body);
});
```

Got is great if you want a smaller library that feels less "bloated" than something like Request.

Final Thoughts

This doesn't cover *all* of the solutions, but now you see how the basic functionality works in a few of the most popular HTTP libraries in Node. There are also libraries such as node-fetch which ports the browser's `fetch` functionality to the backend.

Other languages have a similar variety of libraries to tackle this problem. Check out these other tutorials in Swift, Python and Ruby. Also, check out our Node.js Quickstarts for a place to apply your new skills.

What are your favorite ways to send HTTP requests? Feel free to reach out and let me know or ask any questions:

- Email: Sagnew@twilio.com
- Twitter: @Sagnewshreds
- Github: Sagnew
- Twitch (streaming live code): Sagnewshreds

AUTHORS |  Sam Agnew

Build More With JavaScript

- Twilio Client JavaScript SDK Changelog
- JavaScript Platform Overview
- Build a Video Chat App with JavaScript, Vue.js and Programmable Video
- Get started with writing TypeScript today!
- Accessing Salesforce CRM Data within Twilio Studio

How about the needle ? I like him more and although he has great opportunities for some reason he is not so popular. 😐

Share ›

bfredit • 2 years ago • edited

Please don't specify the version during `npm install` in your blog post. People will blindly copy and paste and will be stuck to outdated version. It's completely unnecessary.

3 Share ›

Bartłomiej Łoszewski ➚ bfredit • 2 years ago

I disagree. If user will copy without thinking than we can't help him. But if for some reason someone will try to use this example in future and sth won't work because of newer version of module than you wouldn't think where did you go wrong but you will see that example was created on old module version. I like showing used module version :)

12 Share ›

Tiago Celestino ➚ Bartłomiej Łoszewski • 2 years ago

I agree.

2 Share ›

bfredit ➚ Bartłomiej Łoszewski • 2 years ago

Yes. Plenty of people just copy and paste. Do you want to give them the finger? Let's be nice and help them be on the latest version.

If your example doesn't work, they'll comment here and everyone will benefit. Or they'll open the GitHub page for the updated API.

If they install an old version unknowingly, you're pushing people to outdated versions — which especially in the case of Request can mean **vulnerabilities**. Then they'll run into issues anyway because you still link to the latest version (which will show a different API). At least be consistent.

Showing the used version makes sense. Having the visitor install it doesn't.

2 • Share ›

Dmitri Pavlutin • 2 years ago • edited

Thanks! Great review.
As an addition, it would be nice to see the difference between bundle sizes.

• Share ›

Josh Yates • 2 years ago

wow, it has become very difficult to keep up with client side development tools...but I'm enjoying the experience.

2 • Share ›

Rachel Lancaster • 2 years ago

nice

• Share ›

Kyle Miller • 2 years ago

If anyone is looking at this article recently, Request has a sister module called request-promise. It is the newest iteration that included promise based Async handling of HTTP reqs

1 • Share ›

POSTS BY STACK

JAVA PHP RUBY PYTHON SWIFT JAVASCRIPT ARDUINO .NET

POSTS BY PRODUCT

SMS AUTHY VOICE MMS VIDEO SIP WIRELESS STUDIO FLEX TASK ROUTER PROGRAMMABLE CHAT TWILIO CLIENT

CATEGORIES

Code, Tutorials and Hacks

Customer Highlights

Developers Drawing The Owl

News

Stories From The Road

The Owl's Nest: Inside Twilio

TWITTER FACEBOOK

Developer stories to your inbox.

Subscribe to the Developer Digest, a monthly dose of all things code.

Enter your email...

You may unsubscribe at any time using the unsubscribe link in the digest email.
See our privacy policy for more information.

NEW!

Tutorials

Sample applications that cover common use cases in a variety of languages. Download, test drive, and tweak them yourself.

Get started

SIGN UP AND START BUILDING

Not ready yet? Talk to an expert.