**NODE CLIENT EXAMPLE**

In a [Node.js](https://nodejs.org/en/about/) app, you can communicate with web APIs by making HTTP requests.

Node.js comes bundled with an [http](https://nodejs.org/api/http.html) and an [https module](https://nodejs.org/api/https.html). These modules have functions to [create an HTTP server](https://www.digitalocean.com/community/tutorials/how-to-create-a-web-server-in-node-js-with-the-http-module) so that a Node.js program can respond to HTTP requests. They can also make HTTP requests to other servers.

In this project, you will use the https module to make HTTP requests to [JSON Placeholder](https://jsonplaceholder.typicode.com/), a fake [REST API](https://en.wikipedia.org/wiki/Representational_state_transfer) for testing purposes. You will begin by making a GET request, the standard HTTP request to receive data. You will then look at ways to customize your request, such as by adding headers. Finally, you will make POST, PUT, and DELETE requests so that you can modify data in an external server.

Let’s get started!

1.

Create a file named getRequest.js— this file will house the code for our application.

2.

To make HTTP requests in Node.js, import the https module.

const https = require('https');

3.

get() function use this format and takes two parameters. The first is the API URL you are making the request to in [string](https://www.digitalocean.com/community/tutorials/how-to-work-with-strings-in-javascript) format and the second is a callback to handle the HTTP response.

https.get(URL\_String, Callback\_Function) { Action }

So write this line of code:

let request = https.get('https://jsonplaceholder.typicode.com/users?\_limit=2', (res) => { });

4.

A successful response would have a 200 status code. The first thing you’ll do in your callback will be to verify that the status code is what you expect.

if (res.statusCode !== 200) {

console.error(`Did not get an OK from the server. Code: ${res.statusCode}`);

res.resume();

return;

}

The res.resume() method tells Node.js to ignore the stream’s data.

5.

Now that you’ve captured error responses, add code to read the data. Node.js responses stream their data in chunks. Inside callback, after if statement;

let data = '';

res.on('data', (chunk) => {

data += chunk;

});

res.on('close', () => {

console.log('Retrieved all data');

console.log(JSON.parse(data));

});

You can store “data” as an array of numbers representing byte data or a string.

6.

Add the following code to capture errors when you’re unable to send an HTTP request:

request.on('error', (err) => {

console.error(`Encountered an error trying to make a request: ${err.message}`);

});

7.

Now execute this program with node getRequest.js

Your console will display this response. This means you’ve successfully made a GET request with the core Node.js library.

### Making Requests with request()

8.

The request() method supports multiple function signatures:

https.request(URL\_String, Options\_Object, Callback\_Function) { Action }

So, create a new JavaScript object that contains a method key:

const options = {

method: 'GET'

};

9.

Now, change get() function name to request().

And add this options object as a second argument of your request function.

10.

Finally, add a end() method for request, before request on() method.

11.

Now execute this program with node index.js. Your console will display this response. You have now used the request() method to make a GET request.

**Well Done!**

12.

In this step, you will use this functionality to configure your request() with the options object.

Remove the URL from the request() call so that the only arguments are the options variable and the callback function.

13.

Now add the following properties to the options object.

host: 'jsonplaceholder.typicode.com',

path: '/users?\_limit=2',

14.

The Accept header specifies the type of data the user can handle. Add the following lines of code to the options object, to append the Accept header:

headers: {

'Accept': 'application/json'

}

15.

Next, run your code once more to make the request by only using options. The results will be the same as your previous runs.

You’ve covered the four most popular options that are sent in Node.js HTTP requests: host, path, method, and headers.

**Nice Work!**

### Step 3 — Making a POST Request

In this section, you’ll create a POST request in Node.js.

You will have to make the following adjustments:

* Change the method in the options object to POST
* Add a header to state you are uploading JSON
* Check the status code to confirm a user was created
* Upload the new user’s data

16.

To make these changes, first create a new file called postRequest.js.

17.

Copy your code from getRequest.js file. And change options object like this:

host: 'jsonplaceholder.typicode.com',

path: '/users',

method: 'POST',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json; charset=UTF-8'

}

18.

Inside the request() function; change status code from “200” to “201”. You may change the console messages.

19.

This POST request is meant to create a new user. For this API, you need to upload the user details. Create some user data and send that with your POST request.

const requestData = { ……… }

You first created the requestData variable, which is a JavaScript object containing user data. Your request should not include an id field, as servers typically generate these while saving the new data.

20.

Next, use the request.write() function, which accepts a string or [object](https://www.digitalocean.com/community/tutorials/using-buffers-in-node-js) to send along with the request. As your requestData variable is an object, you should use the JSON.stringify function to convert it to a string.

request.write(JSON.stringify(requestData));

NOTE: It’s important that you write data before you use the end() function. The end() function tells Node.js that there’s no more data to be added to the request and sends it

### Step 4 — Making a PUT Request

21.

Next As the code is similar to the POST request, you’ll use that module as a base for this one. Copy the postRequest.js into a new file, putRequest.js

22.

Make these changes inside options, so that you send a PUT request.

path: '/users/1',

method: 'PUT',

23.

Change status code to “200”. You may change the console messages also.

24.

Modify your object, on which data you want to update.

const requestData = {

username: 'doejohn'

};

25.

Now execute this Node.js program in your terminal. You will receive your updated data.

You have just sent a PUT request to update a pre-existing user.

### Step 4 — Making a DELETE Request

26.

The code you will write is similar to that of a GET request, so use that module as a base for this one. Copy the getRequest.js file into a new deleteRequest.js file.

27.

Make these changes inside options, so that you send a DELETE request.

path: '/users/1',

method: 'DELETE',

28.

Make Run this module to confirm it works.

While the API does not return a response body, you still got a 200 response so the request was OK.

You’ve now learned how to make DELETE requests with Node.js core modules.