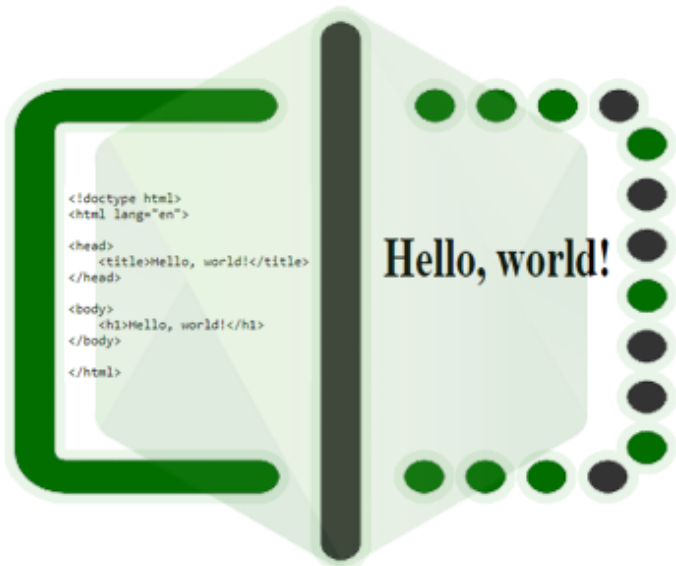


Render HTML In Node.js

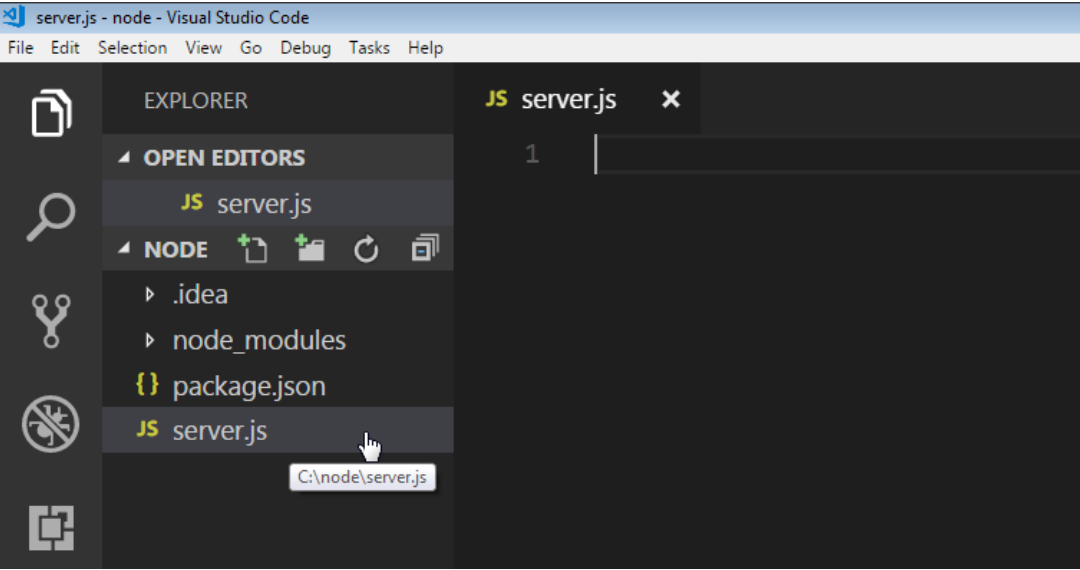


Let’s see how to create a server and render some HTML in Node.js. First up, we will create a

new file named `server.js` in our project root. We are writing our first server in Node.js! It might seem odd if you’re coming from a PHP background to talk about creating our first server in a JavaScript file, but that is in fact what we are about to do. This is because we are used to having a server like Nginx or Apache which are set up to interpret PHP code. The Nginx or Apache process then serves the HTML files to the user. In this case we are writing our own server. Let’s see how to complete this task.

Creating A Web Server In Node.js




Node.js provides the ability to [create server functionality](#) and bypass the traditional idea of a stand alone web server. For example in Node.js, we can specify a port to communicate on, which domain to use, and now to handle http requests. This new file will have instructions to listen to http requests, and do something with these requests.





This `server.js` file is the file that Node.js will execute, and will set up a type of loop. Node will continue to listen to requests as long as we allow the program to run. Let’s add


DEDICATED SERVER AGILE S


1 GBPS BANDWIDTH


   DELIVERED IN 1H

 1 CPU (4C/4T) @3 GHZ

 GEFORCE GT 710 1 GB

 1 TB SATA 3 ou 240 GB SSD

 8 GB RAM DDR4

 €29⁹⁹ / MTH .EX. VAT

DISCOVER

www.ikoula.com

Advertise Here



[WordPress Dashboard](#)

[Tutorial](#)



[Build A Link Sharing Website](#)

[With Laravel](#)



[MySQL Group By Having Limit](#)

[Offset and More!](#)



[29 Awesome JavaScript](#)

[Learning Tutorials](#)

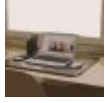


[Making Favorites Part Of The](#)

[Activity Feed](#)



[Default Function Parameters In ES6](#)



[PHP Code Structure](#)



[Build Your Own PCRE Regex Tester](#)

[With PHP](#)



[How To Toggle Exception](#)

[Handling](#)



[The Ultimate Guide to PHP](#)

[Functions](#)



[ES6 Promises Tutorial](#)

a bit of code to our file.

server.js

```
1 let http = require('http');
2
3 let handleRequest = (request, response) => {
4   response.writeHead(200, {
5     'Content-Type': 'text/plain'
6   });
7   response.write('Hi There!');
8   response.end();
9 };
10
11 http.createServer(handleRequest).listen(8000);
```

So what does this file do? Well, we can see that we use **require()** like we've learned about before to make use of the **http** module in this file. Once the http module is imported, we have access to the **createServer()** method. Believe it or not, that one method creates an entire web server for you! Notice that we also chain on the **listen()** method which specifies which port to listen for http requests on. Now, we see that `createServer()` is actually accepting an argument of `handleRequest`. This is required because it is in this function that we set up the logic to handle any http requests the server gets. We created this function ourselves, and then pass in a reference to that function.

So what is this `handleRequest()` function doing? Well we can see that it accepts a `request` and a `response` as two arguments. These arguments are automatically passed in to this function by Node.js. So in this function, we are not really going to do anything with the request, but we do need to set up what the server will do with a response. We first use the **writeHead()** method to set the header of the response. The first argument to that function is the status code we would like to send. A `200` means everything went well. The next argument that gets passed is a JavaScript object which specifies the content type we are sending back. After that we call the **write()** function, so that we can just render some text to the screen. Lastly, we call the **end()** method to indicate that we are done handling the request and the response can be sent to the user who made the http request.

Let's try it out, we can run our program by typing `node server.js` at the command line and notice that it looks like the program is just hanging. Well, it is basically running in a loop so that it can respond any time a request comes in.



[jQuery Selectors and Filters](#)



[D3 DOM Selection With D3 JavaScript](#)



[Many to Many Relationships in Laravel](#)



[Higher Order Functions In JavaScript](#)



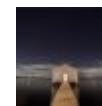
[How To Refactor Code To A Dedicated Class](#)



[VueJS Dynamic Components](#)



[Vue.js for Interactive Web Interfaces](#)



[Linux Files and Directories](#)



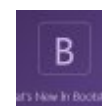
[Angular Pipes And Data Binding](#)



[25 Popular Applications Built With Laravel](#)



[How To Use The Find Function In Underscore JS](#)



[What Is New In Bootstrap 4](#)



[Underscore JS sortBy Function](#)



[Most Popular JavaScript Frameworks](#)



[Vue.js Express Tutorial](#)



[Simple Styling Techniques For React Elements](#)



[How To Display API Data Using](#)

```
c:\node>node server.js
```

Now, we can load up `http://localhost:8000/` in the browser and see what happens.

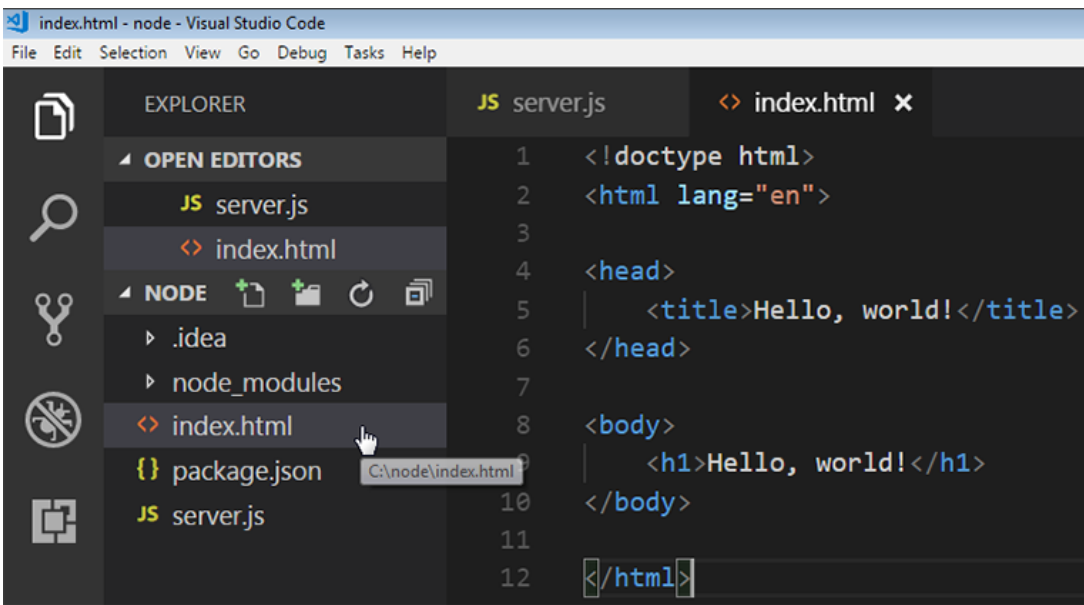


Cool! We just set up a fully functioning web server in Node.js in about 10 lines of code! Here are the important methods we used.

- [http.createServer\(\)](#)
- [.listen\(\)](#)
- [response.writeHead\(\)](#)
- [response.write\(\);](#)
- [response.end\(\)](#)

Serving HTML Files With Node.js

Now that we have a server up and running, let's see how to [render html with Node.js](#). To do this, we of course need an html file to serve so we can create and populate an `index.html` file now.



index.html

```
1 <!doctype html>
2 <html lang="en">
3
4 <head>
5   <title>Hello, world!</title>
6 </head>
7
8 <body>
9   <h1>Hello, world!</h1>
10 </body>
11
12 </html>
```

React.js



[Underscore JS
Some Function](#)



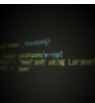
[How To Create
Custom Templates
For WordPress Pages](#)



[PHP Flow Control
Statements](#)



[Angular Table
Filter Component](#)



[Send a Tweet with
Laravel](#)



[Install Larabook
On Laravel
Homestead](#)



[What a Constant is
and how to
declare one](#)



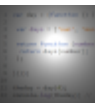
[What is
WordPress?](#)



[C# Control
Statements](#)



[Douglas Crockford
The Good Parts
Examples](#)



[JavaScript
Functions Tutorial](#)



[Learn PHP and
Programming by
using the Eclipse PDT
Debugger](#)



[Interface
Examples For
Object Oriented PHP](#)



[Flash Messages in
Laravel](#)



[PHP Variables and
Strings Tutorial](#)



[PHP MySQL CRUD
Tutorial](#)



[Top 12 Websites
for Twitter](#)

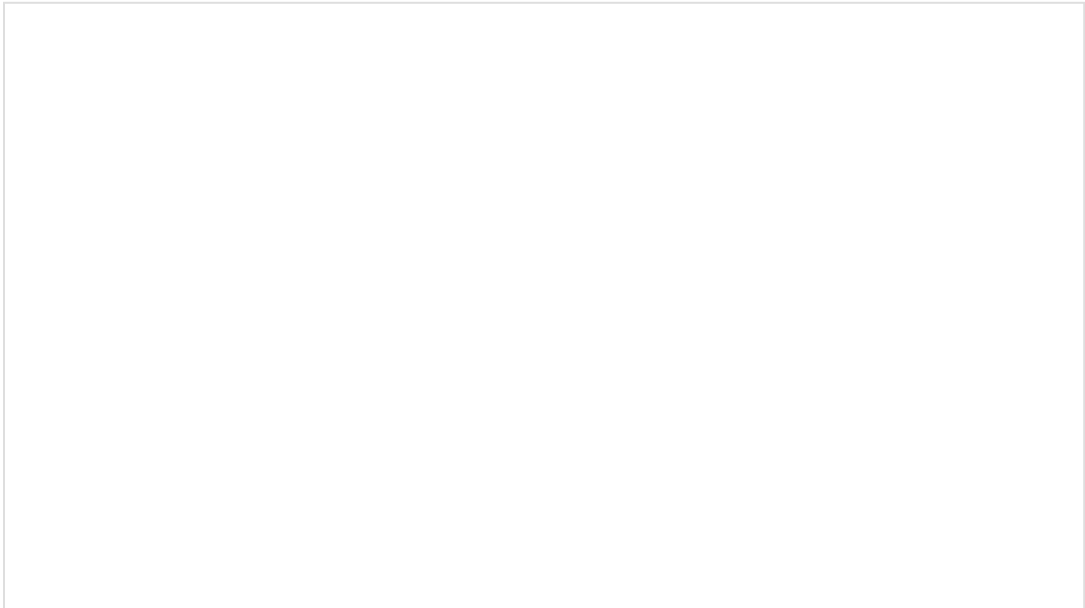
So in the section above, we simply sent some text to the browser in response to an incoming http request. Now, we no longer want to send just simple text, but rather an actual html file. How can we do this? The index.html file is currently on our server. We would need to somehow grab the file and attach it to a response, and then send that response back to the user who made the http request. Therefore we need a way to use the file system to get the file, and then send it to the user.

Fetching The HTML File on our File System

In the snippet below, the first addition we will make to our server.js script is to require the file system module in Node.js. You’re likely a pro at that by now! Then in the original response.writeHead() method, notice that we change the content type from ‘text/plain’ to ‘text/html’. This changes the header that gets sent to the browser and informs the browser that it is now dealing with an actual html file and to parse it as such. Once that is in place, we use the file system module to make a call to [fs.readFile\(\)](#) and we read in the index.html file that lives on our server. We specify the file we want to read in with the first argument. We don’t need the second argument in this case so we set it to null, but we do need the third argument which is a callback function. This function is executed once Node.js has finished reading in the index.html file. In other words, it is a way to tell Node.js what to do next so to speak.

Attaching the file to the response and sending it to the browser

What we use this callback function for is to check to make sure there are no errors, but if there is an error – we will send back a 404 not found message to the browser. However, if the operation is successful, we are going to send a response back to the browser and we are attaching the file to that response by passing it as an argument. This data is our actual index.html file. Finally, we make a call to response.end()).



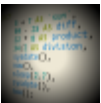
Bootstrap



[Flarum Forum Software](#)



[CSS Grid Tutorial](#)



[MySQL Function Tutorial](#)



[JavaScript Callbacks vs Promises vs Async Await](#)



[Twitter Bootstrap Grid Tutorial](#)



[Vuejs Form Example](#)



[Liskov Substitution Principle](#)



[VueJS Textarea Binding](#)



[Mentions And Notifications](#)



[Build Your Own News Aggregator with Twitter Bootstrap and SimplePie](#)



[3 Examples of the Underscore Reduce Function](#)



[Autoloading For Code Organization](#)



[jQuery AJAX Tutorial](#)



[Command Types In Linux](#)



[Mithril JavaScript Tutorial](#)



[Combine PHP Functions To Make Your Own](#)



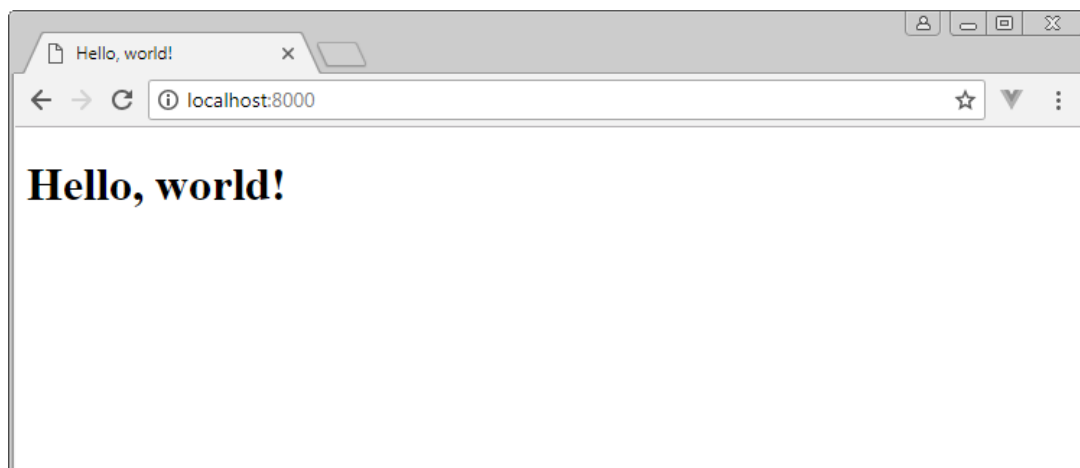
[Vuejs Router Tutorial](#)

```
1 let http = require('http');
2 let fs = require('fs');
3
4 let handleRequest = (request, response) => {
5   response.writeHead(200, {
6     'Content-Type': 'text/html'
7   });
8   fs.readFile('./index.html', null, function (error, data) {
9     if (error) {
10      response.writeHead(404);
11      response.write('Whoops! File not found!');
12    } else {
13      response.write(data);
14    }
15    response.end();
16  });
17 };
18
19 http.createServer(handleRequest).listen(8000);
```

In the meantime, make sure your server is running with nodemon by typing `nodemon server.js` at the command prompt. We can see how the server automatically restarts as we edit and update our JavaScript code.

```
c:\node>nodemon server.js
[nodemon] 1.17.4
[nodemon] to restart at any time, enter rs
[nodemon] watching: *.*
[nodemon] starting node server.js
[nodemon] restarting due to changes...
[nodemon] starting node server.js
[nodemon] restarting due to changes...
[nodemon] starting node server.js
[nodemon] restarting due to changes...
[nodemon] starting node server.js
```

Loading up the browser at `http://localhost:8000/` shows us that now, we are getting a glorious html rendering of our `index.html` file!



Just for grins, let's see what happens if we had not changed the content type.

Laravel Subscription



System Tutorial



Angular Routing Example



The 27 Most Popular File Functions in PHP

Functions in PHP



Single Responsibility Principle

Principle



How To Make HTTP Requests In Angular Using Observables



CSS Flexbox Tutorial



How To Send Email To New Users

Users



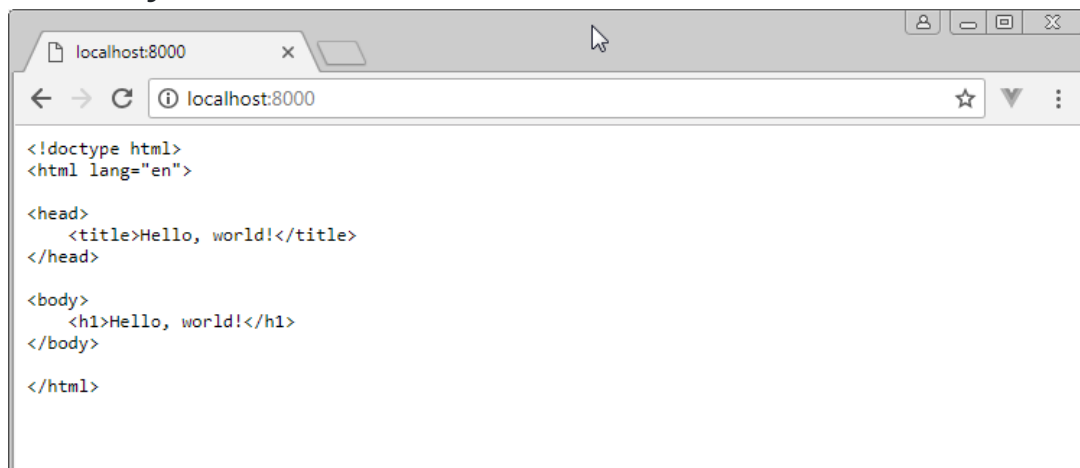
How To Use wp_list_pages() To Create Parent and Child Page Menus



Dependency Injection for Beginners

```
1 let http = require('http');
2 let fs = require('fs');
3
4 let handleRequest = (request, response) => {
5   response.writeHead(200, {
6     'Content-Type': 'text/plain'
7   });
8   fs.readFile('./index.html', null, function (error, data) {
9     if (error) {
10      response.writeHead(404);
11      response.write('file not found');
12    } else {
13      response.write(data);
14    }
15    response.end();
16  });
17 };
18
19 http.createServer(handleRequest).listen(8000);
```

Loading the page now gives us a very different result. Now we see how important it is to [set the response header](#) correctly!



Render HTML In Node.js Summary

In this tutorial, we saw how to output html files to the browser with Node.js. It was pretty cool to see how we were able to create our own web server with just a very minimal amount of JavaScript code. We also saw how to read in a file from the file system, attach it to a response, and send it back to the user. Of course we'll be looking at how to vastly simplify all of this using some of the [Popular Node.js Frameworks](#) soon, but for now it's great to see how to make things happen using raw JavaScript and the Node.js runtime only.

[#javascript](#)[#nodejs](#)

