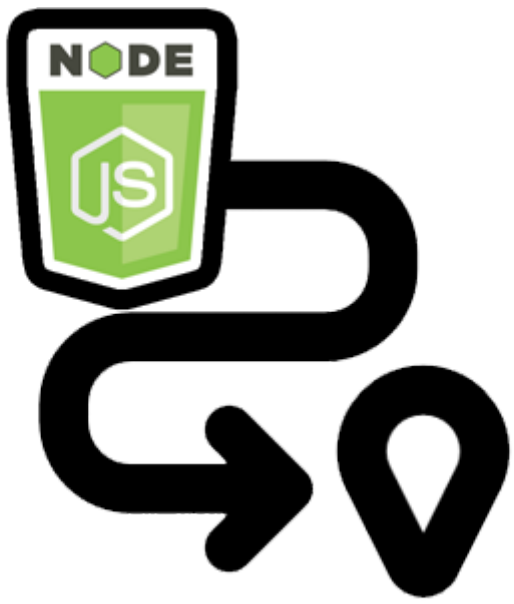
← Render HTML In Node.js | Express.js Beginner Tutorial →

Node.js Routes Tutorial



In our last Node.js tutorial, we learned how to render some very basic html to the screen. Of course an actual website or application will have many different html files to present to the user. Typically you will need some type of routing mechanism to handle this. In this episode, we'll take a look at a routes example where a user can load different urls in the browser. Each different url will be a different route, which in turn will load a different html page from the node.js server. Let's see how we can set up very basic Node.js routes now.

Two Different HTML Files

We need to have at least two html files to demonstrate routing in Node.js. We will stick with the simple index.html file from the last lesson, and also we will create a new about.html file in the project. Here are those two files if you are following along.

index.html

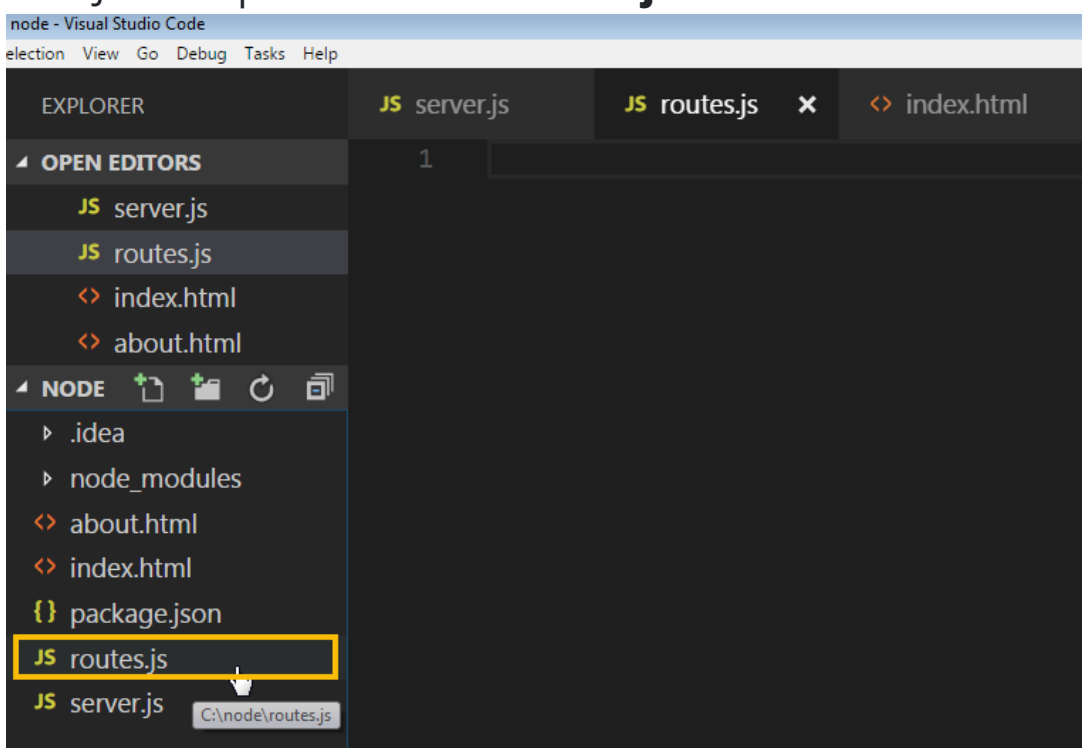
```
<!doctype html>
<html lang="en">

<head>
    <title>Hello, world!</title>
</head>

<body>
    <h1>Hello, world!</h1>
</body>

</html>
```

about.html

```
<!doctype html>
<html lang="en">

<head>
    <title>About Us!</title>
</head>

<body>
    <h1>Hiya Friends! This is an html file that is about
</body>

</html>
```

Putting routes in a separate file

Often times in Node.js, there will be a **separate file** which will contain all the routes for the application. This JavaScript file will handle the incoming http requests and assign correct routes as needed. As such, let's create a new JavaScript file named **routes.js**.



Twitter Bootstrap 12 Column Grid
C# Classes For A CRM Application
JavaScript Functions Tutorial
How Do Linux Permissions Work?
Render HTML In Node.js
What are Laravel Filters?
Angular Pipes And Data Binding
How To Add Search To A WordPress Theme
Laravel Collections Tutorial
Mongoose Validation Examples
Underscore JS Map Function
PHP Cookies And Sessions
The Ultimate Guide To Object Oriented PHP
MySQL Function Tutorial
You Might Still Need jQuery
Embedding Expressions In JSX
Mentions And Notifications
Upgrading VueJS

So what are we going to put in this routes.js file? Well, what we need to setup is an export of a function which we can then use in the main server.js file. We can start with this.

routes.js

```
module.exports = {
    handleRequest(request, response) {
        response.writeHead(200, {
            'Content-Type': 'text/html'
        });
    }
}
```

Using the url module

In order to set up routing, the application needs to be aware of what url was typed into the browser. Node.js has a **url module** which allows us split up a url into all of it's **readable parts**. We are going to needs this to set up our routing so let's include it in our routes.js file now.

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

module.exports = {
    handleRequest(request, response) {
        response.writeHead(200, {
            'Content-Type': 'text/html'
        });
    }
}
```

Configuring Our Two Routes

Now we are ready to set up a couple of routes. One route will load the **index.html** page and the other roue will load the **about.html** file. The way that we do this is to first determine the path. In the case of finding a path of `/`, then index.html will get rendered. In the case of a path of `/about.html`, then about.html will get rendered. Let's update the module.exports in routes.js to the following code.

VueJS Subnet Calculator

Single File Components In VueJS

The 27 Most Useful PHP Array Functions You Need To Know!

What Are PHP Arrays?

PHP String Helper Functions

How To Quickly Test PHP Snippets

How To Refactor Code To A Dedicated Class

How To Fix The N+1 Problem

WordPress Toolbar Tutorial

jQuery Effects and Animation Methods

Laravel Routing Tutorial

JavaScript Tutorials For Beginners

Laravel Testing Helpers

Laravel Aliases and Contracts

Vi Editor Tutorial For Beginners

Digging In To HTML Fundamentals

```
module.exports = {
    handleRequest(request, response) {
        response.writeHead(200, {
            'Content-Type': 'text/html'
        });

        let path = url.parse(request.url).pathname;

        switch (path) {
            case '/':
                html.render('./index.html', response);
                break;
            case '/about':
                html.render('./about.html', response);
                break;
            default:
                response.writeHead(404);
                response.write('Route not found');
                response.end();
        }
    }
}
```

In this code above, we first find the path we are interested in via **url.parse()** and **pathname**. Very nice.

Using a `switch` statement to route

Now that we have the path, we can use a **switch statement** to load different html files based on what the path variable holds. We can see that if the path holds the value of '/', then a render function is called passing the './index.html' file as an argument. On the other hand, if the path holds the value of '/about.html' then that same render function would fire but this time it would take the string of './about.html' as an argument. Lastly, we set up the default scenario which provides for any route that does not exist. Notice that we use an **ES6 Object Literal** to store that `render()` function we need. Here is our full routes.js file now.

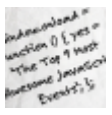
routes.js


Adding Game Reviews With Eloquent Relationships


Escape Strings For MySQL To Avoid SQL Injection


Developing With VueJS and PHP


WordPress Links and Images


JavaScript Events Tutorial


Twitter Bootstrap Navigation Elements


Underscore JS Each Function


Learning About Regular Expressions


What Is A WordPress Archive Page?


Node.js Todo List Tutorial


Setting Permissions With Policy Objects


What Is Goutte?


How To Compare Two Arrays of Data and Calculate Position Differences

Twitter Bootstrap Modal Tutorial


jQuery AJAX Tutorial


Applying RESTful Methods to the Reviews Resource

```
const url = require('url');
let fs = require('fs');

html = {
    render(path, response) {
        fs.readFile(path, null, function (error, data) {
            if (error) {
                response.writeHead(404);
                respone.write('file not found');
            } else {
                response.write(data);
            }
            response.end();
        });
    }
}

module.exports = {
    handleRequest(request, response) {
        response.writeHead(200, {
            'Content-Type': 'text/html'
        });

        let path = url.parse(request.url).pathname;

        switch (path) {
            case '/':
                html.render('./index.html', response);
                break;
            case '/about':
                html.render('./about.html', response);
                break;
            default:
                response.writeHead(404);
                response.write('Route not found');
                response.end();
        }
    }
}
```

Importing routes.js to server.js

The final thing we'll need to do is to make sure that we are importing routes.js into our main server.js file.

server.js

How Do Functions Work in Python?

Build A Regular Expression Tester With Laravel

Send Email With Laravel

What Is New In Bootstrap 4

jQuery Event Handling

Check Authorization With Policies Before Delete Function

How To Protect Specific Routes With Middleware

Creating Static And Dynamic Web Pages In Laravel

Working With HTML Images

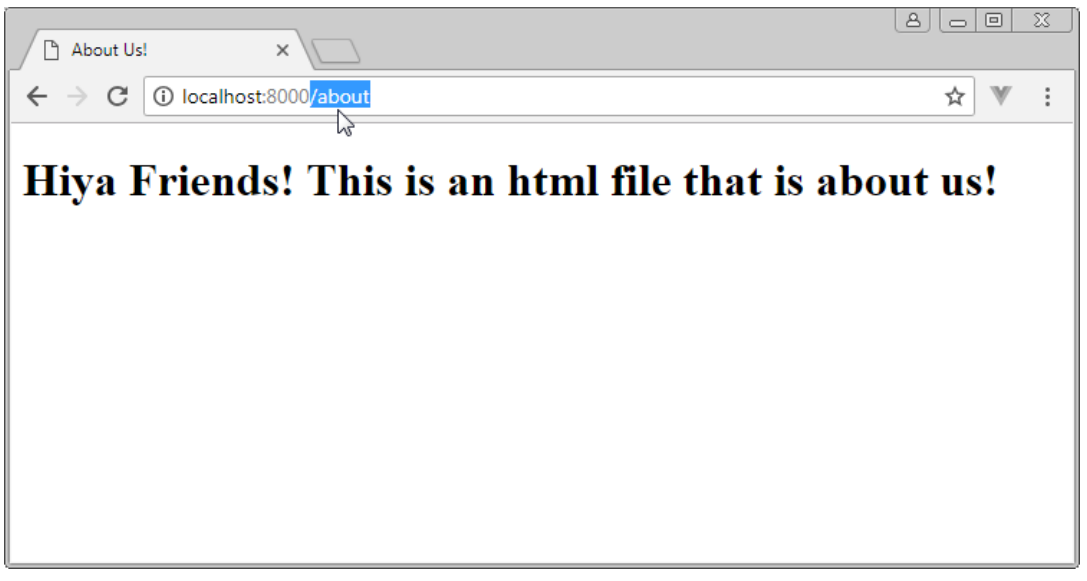
```
let http = require('http');
let router = require('./routes');

let handleRequest = (request, response) => {
    response.writeHead(200, {
        'Content-Type': 'text/html'
    });

};

http.createServer(router.handleRequest).listen(8000);
```

In the snippet above, we first see that we use *let router = require('./routes');* to store all the logic we just built in routes.js right into that `router` variable. Now, when we go ahead and create the server, we can make use of the `router.handleRequest` function to actually process the routes. Pretty cool. Let's fire up our server and test it out in the browser.

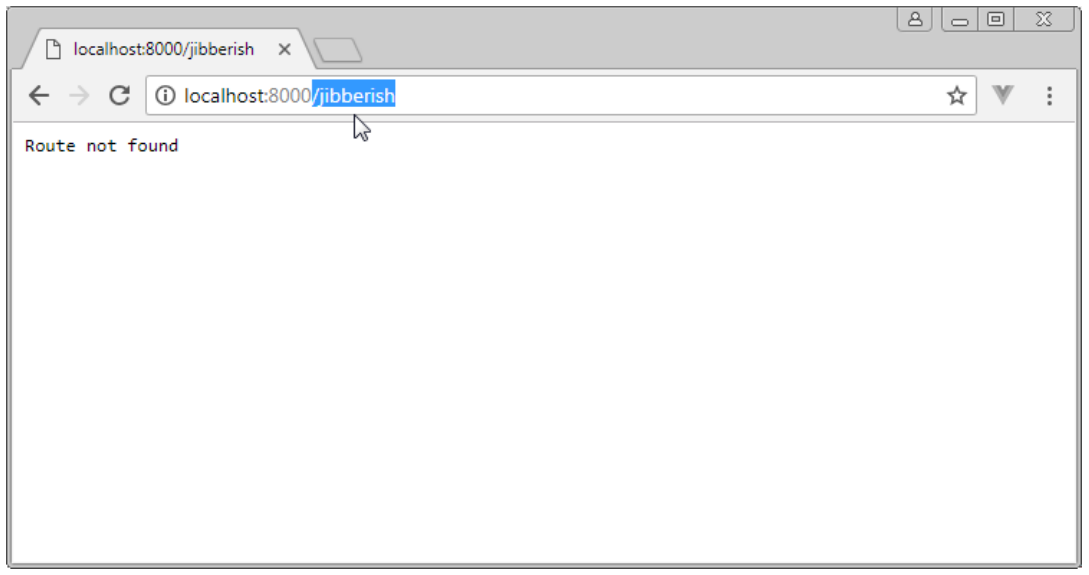
```
c:\node>node server.js
```

Visiting http://localhost:8000/

Visiting http://localhost:8000/about



We can even try visiting a jibberish route like this http://localhost:8000/jibberish, and because we set up our routing to handle this we will get a nice error message.



Node.js Routes Tutorial Summary

In this tutorial, we learned how to set up routes in Node.js from scratch. This helps us to understand the basics of how Node.js works. In almost all cases, you would not actually have to set up your own routing in Node because you could use one of the many available frameworks that do all the hard work for you. For example **Express.js** could be used to vastly simplify what we created above. It is still helpful however to see how to do it the long way so to speak, before using a framework to make things easier.

#javascript #nodejs

← Render HTML In Node.js | Express.js Beginner Tutorial →