**Node JS - Router and Routes** - [Sjlouji](https://medium.com/@sjlouji10?source=post_page-----a4a3cfced5c4--------------------------------) Sep 12, 2020

Hello all. In this blog, I am explaining how to perform routing with Node JS. Routing is one of the most important parts of any Web framework since it defines how our application should handle all the HTTP requests by the client.

**Table of Content**

1. [Create a Node JS Projec](https://medium.com/swlh/node-js-router-and-routes-a4a3cfced5c4#d64f)t
2. [Creating a simple Server with Express JS](https://medium.com/swlh/node-js-router-and-routes-a4a3cfced5c4#d002)
3. [Organizing Routes in the Main File](https://medium.com/swlh/node-js-router-and-routes-a4a3cfced5c4#952d)
4. [Organizing all routes in a Separate file](https://medium.com/swlh/node-js-router-and-routes-a4a3cfced5c4#2502)
5. [Installing Logger Middleware — Morgan](http://eca5/)
6. **Creating a Node JS Project**

Create a new directory and initialize node with the command npm init.

mkdir helloworld  
cd helloworld/  
npm init -y

After executing the command, a package.json file generated in the project’s root directory. This holds all the metadata relevant to the project.

On this file, we see something called scripts. This is the place where we add our own commands for the project. I am creating a new command which starts my server when I type npm start. The script tells node that it should run the command node index.js every time when I execute the command npm start.

package.json

"scripts": {  
 **"start": "node index.js",**  
 "test": "echo \"Error: no test specified\" && exit 1"  
},

2. **Creating a simple Server with Express JS**

Now let’s create our server. Here we are creating our server using Express.js. Express JS is an open-source web framework for node JS. It is designing for building web apps and APIs. The below command installs express to our project.

npm install express --save

[Express - Node.js web application framework](https://expressjs.com/" \t "_blank)

[Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and…](https://expressjs.com/" \t "_blank)

[expressjs.com](https://expressjs.com/" \t "_blank)

We are using express to create a new server that will be running on the port 8000. Also for the demonstration, I am creating a route that returns hello world.

index.js

**var express = require('express');  
var app = express();**app.get('/', function (req, res) {  
 res.send('Hello World!');  
});**app.listen(8000, function () {  
 console.log('Listening to Port 8000');  
});**

Now start the server, you should see hello world being displayed in the browser.

npm start

3. **Organizing Routes in the Main File**

We have successfully created a server and a basic route. Now let’s begin with creating some more routes as done below.

index.js

var express = require('express');  
var app = express();**app.get('/', function (req, res) {  
 res.send('Hello Joan!');  
});app.get('/settings', function (req, res) {  
 res.send('Settings Page');  
});app.get('/settings/profile', function (req, res) {  
 res.send('Profile Page');  
});app.get('/blogs', function (req, res) {  
 res.send('All Blogs');  
});app.get('/blogs/:id', function (req, res) {  
 res.send('View Blogs');  
});**app.listen(8000, function () {  
 console.log('Listening to Port 8000');  
});

Once done, visit all the routes. You should see all the responses of respective routes being displayed on the browser.

Graphical user interface, text, application

Description automatically generated

Snapshot of route /settings/profile

4. **Organizing all routes in a Separate file**

Declaring all the routes in the main file is not a good practice. When there is a situation where we are creating 1000s of routes, it makes our main file look messy, and also debugging becomes so tough. In order to get rid of this problem, we can write all these routes in a separate file and just call them in the main file.

As seen above, we have created three main routes for settings, blogs, and dashboard. So I am creating three javascript files named settings.js, blogs.js, dashboard.jsunder a newly created Routes folder. My project structure looks like this.

Graphical user interface, text, application

Description automatically generated with medium confidence

Project Structure

The dashboard.js file handles all the routes with a prefix /.

Blogs.js file handles all the routes with a prefix /blogs.

Settings.js file handles all the routes with a prefix /settings.

Routes/blogs.js

const express = require('express')  
let app = express.Router()**app.get('/', function (req, res) {  
 res.send('All Blogs');  
});app.get('/:id', function (req, res) {  
 res.send('View Blogs' + req.params.id);  
});**module.exports = app

Routes/dashboard.js

const express = require('express')  
let app = express.Router()**app.get('/', function (req, res) {  
 res.send('Dashboard Page');  
});**module.exports = app

Routes/settings.js

const express = require('express')  
let app = express.Router()**app.get('/', function (req, res) {  
 res.send('Settings Page');  
});app.get('/profile', function (req, res) {  
 res.send('Profile Page');  
});**module.exports = app

After creating these files, I am just using them as an express middleware in my main file which is our index.js file. Now just start the server and check if it still works fine. It will.

index.js

var express = require('express');  
var app = express();  
**const settings = require('./Routes/settings')  
const dashborad = require('./Routes/dashboard')  
const blogs = require('./Routes/blogs')app.use('/',dashborad)  
app.use('/blogs',blogs)  
app.use('/settings',settings)**app.listen(8000, function () {  
 console.log('Listening to Port 8000');  
});

Graphical user interface, text, application

Description automatically generated

Routes Example

Without separating the routes, we need to call routers with the same prefix again and again. Like, If two routers have the same prefix as settings, we need to call both the routers with the prefix name settings again and again. It will be good if we are using only two routers. But it is not the same everywhere right. In a real-world application, we will be using 1000s of routes. This is the reason why we are separating the routes.

5.**Installing Logger Middleware — Morgan**

In other frameworks like Django, Laravel, we might have seen all the requests and responses we make will be logged in the terminal. So that we can easily identify if our request is valid or has any error. But till now, our server hasn’t logged anything in the terminal. There are some traditional ways to log all the requests. But to make it simple, I am using an express middleware called **Morgan.**

[morgan](https://expressjs.com/en/resources/middleware/morgan.html" \t "_blank)

[HTTP request logger middleware for node.js Named after Dexter, a show you should not watch until completion. This is a…](https://expressjs.com/en/resources/middleware/morgan.html" \t "_blank)

[expressjs.com](https://expressjs.com/en/resources/middleware/morgan.html" \t "_blank)

**Morgan** is a middleware that allows us to easily log requests, errors, and more to the console. It’s easy to **use**, but still powerful and customizable.

npm install morgan

Configuring this is so simple. Just pass morgan within app.use().

index.js

**var morgan = require('morgan')app.use(morgan('combined'))**  
app.use('/',dashborad)  
app.use('/blogs',blogs)  
app.use('/settings',settings)

Now run the server, so that all the requests will be consoled in the terminal.



Logging all request with Morgan

[Do check my previous blog, where I have explained the basics of Node.js](https://medium.com/@sjlouji10/node-js-hello-world-with-node-js-and-express-61dd390c0b14).

Feel free to contact me for any queries.

Email: sjlouji10@gmail.com

Linkedin: <https://www.linkedin.com/in/sjlouji/>

Complete code on my GitHub: <https://github.com/sjlouji/Node-JS-Basics---Medium>