# **Setting up Node.js and NPM**

Note: Make sure you have installed Git on your machine before you install Node.js. Please complete the previous Git installation exercise before proceeding with this exercise.

### **Objectives and Outcomes**

In this exercise, you will learn to set up the Node.js environment, a popular Javascript based server framework, and node package manager (NPM) on your machine. To learn more about NodeJS, you can visit [https://nodejs.org](https://nodejs.org/). For this course, you need to install Node.js on your machine. At the end of this exercise, you will be able to:

* Complete the set up of Node.js and NPM on your machine
* Verify that the installation was successful and your machine is ready for using Node.js and NPM.

### **Verifying the Node Installation**

* Open a terminal window on your machine. If you are using a Windows machine, open a cmd window or PowerShell window with admin privileges.
* To ensure that your NodeJS setup is working correctly, type the following at the command prompt to check for the version of Node and NPM

node -v

npm -v

# **Understanding Node Modules**

### **Objectives and Outcomes**

In this exercise, you will learn about writing Node applications using JavaScript and also learn about the basics of Node modules. At the end of this exercise, you will be able to:

* Write a simple Node application in JavaScript.
* Understand the basics of Node modules and write simple file-based Node modules

### **Starting a Node Application**

* Go to a convenient location on your computer and create a folder named *NodeJS*. Then move to this folder.
* Now create a folder named *node-examples* and then move into this folder.
* At the prompt, type the following to initialize a package.json file in the node-examples folder:

npm init

* Accept the standard defaults suggested and then update the *package.json* file until you end up with the file containing the following:

{

"name": "node-examples",

"version": "1.0.0",

"description": "Simple Node Examples",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node index"

},

"author": "Jogesh Muppala",

"license": "ISC"

}

* Create a file named index*.js* and add the following code to this file:

var rect = {

perimeter: (x, y) => (2\*(x+y)),

area: (x, y) => (x\*y)

};

function solveRect(l,b) {

console.log("Solving for rectangle with l = " + l + " and b = " + b);

if (l <= 0 || b <= 0) {

console.log("Rectangle dimensions should be greater than zero: l = "

+ l + ", and b = " + b);

}

else {

console.log("The area of the rectangle is " + rect.area(l,b));

console.log("The perimeter of the rectangle is " + rect.perimeter(l,b));

}

}

solveRect(2,4);

solveRect(3,5);

solveRect(0,5);

solveRect(-3,5);

* To run the Node application, type the following at the prompt:

npm start

* To initialize a Git repository and add the current files in the folder to the repository, type the following at the prompt:

git init

git add .

* Then, do a Git commit with the message "Simple Node Example".

### **A Simple Node Module**

* Now, create a file named *rectangle.js*, and add the following code to it:

exports.perimeter = (x, y) => (2\*(x+y));

exports.area = (x, y) => (x\*y);

* Then, update *index.js* as follows:

var rect = require('./rectangle');

. . .

* Run the Node application like before and observe that the result will be the same.
* Do a Git commit with the message "Simple Node Module".

### 

### **Conclusions**

In this exercise, you learnt about writing simple Node applications in JavaScript. Thereafter you learn about writing a simple Node module and use it within your Node application.

**Callbacks and Error Handling**

### **Objectives and Outcomes**

In this exercise, you will learn about callbacks, JavaScript closures and error handling in Node applications. At the end of this exercise, you will be able to:

* Using Callbacks in Node applications
* Error handling in Node applications

### **Using Callbacks and Error Handling**

* Update *rectangle.js* as shown below:

module.exports = (x,y,callback) => {

if (x <= 0 || y <= 0)

setTimeout(() =>

callback(new Error("Rectangle dimensions should be greater than zero: l = "

+ x + ", and b = " + y),

null),

2000);

else

setTimeout(() =>

callback(null, {

perimeter: () => (2\*(x+y)),

area:() => (x\*y)

}),

2000);

}

* Then, update *index.js* as shown below:

. . .

function solveRect(l,b) {

console.log("Solving for rectangle with l = "

+ l + " and b = " + b);

rect(l,b, (err,rectangle) => {

if (err) {

console.log("ERROR: ", err.message);

}

else {

console.log("The area of the rectangle of dimensions l = "

+ l + " and b = " + b + " is " + rectangle.area());

console.log("The perimeter of the rectangle of dimensions l = "

+ l + " and b = " + b + " is " + rectangle.perimeter());

}

});

console.log("This statement after the call to rect()");

};

. . .

* Run the Node application as before and see the result.
* Do a Git commit with the message "Node Callbacks and Error Handling".

### **Conclusions**

In this exercise, you learnt about using Callbacks and error handling in Node applications. In addition you learnt about using external Node modules.

# 

# **Node and the HTTP Module**

### **Objectives and Outcomes**

In this exercise, you will explore three cores Node modules: HTTP, fs and path. At the end of this exercise, you will be able to:

* Implement a simple HTTP Server
* Implement a server that returns html files from a folder

### **A Simple HTTP Server**

* Create a folder named *node-http* in the *NodeJS* folder and move into the folder.
* In the *node-http* folder, create a subfolder named *public*.
* At the prompt, type the following to initialize a package.json file in the node-examples folder:

npm init

* Accept the standard defaults suggested until you end up with a package.json file containing the following:

}

* Create a file named index*.js* and add the following code to it:

const http = require('http');

const hostname = 'localhost';

const port = 3000;

const server = http.createServer((req, res) => {

console.log(req.headers);

res.statusCode = 200;

res.setHeader('Content-Type', 'text/html');

res.end('<html><body><h1>Hello, World!</h1></body></html>');

})

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

* Start the server by typing the following at the prompt:

npm start

* Then you can type [http://localhost:3000](http://localhost:3000/) in your browser address bar and see the result.
* You can also use [postman](http://www.getpostman.com/) chrome extension to send requests to the server and see the response. Alternately, you can download the stand-alone Postman tool from [http://getpostman.com](http://getpostman.com/) and install it on your computer.
* Initialize a Git repository, check in the files and do a Git commit with the message "Node HTTP Example 1".

### **Serving HTML Files**

* In the *public* folder, create a file named *index.html* and add the following code to it:

<html>

<title>This is index.html</title>

<body>

<h1>Index.html</h1>

<p>This is the contents of this file</p>

</body>

</html>

* Similarly create an aboutus.html file and add the following code to it:

<html>

<title>This is aboutus.html</title>

<body>

<h1>Aboutus.html</h1>

<p>This is the contents of the aboutus.html file</p>

</body>

</html>

* Then update *index.js* as follows:

. . .

const fs = require('fs');

const path = require('path');

. . .

const server = http.createServer((req, res) => {

console.log('Request for ' + req.url + ' by method ' + req.method);

if (req.method == 'GET') {

var fileUrl;

if (req.url == '/') fileUrl = '/index.html';

else fileUrl = req.url;

var filePath = path.resolve('./public'+fileUrl);

const fileExt = path.extname(filePath);

if (fileExt == '.html') {

fs.exists(filePath, (exists) => {

if (!exists) {

res.statusCode = 404;

res.setHeader('Content-Type', 'text/html');

res.end('<html><body><h1>Error 404: ' + fileUrl +

' not found</h1></body></html>');

return;

}

res.statusCode = 200;

res.setHeader('Content-Type', 'text/html');

fs.createReadStream(filePath).pipe(res);

});

}

else {

res.statusCode = 404;

res.setHeader('Content-Type', 'text/html');

res.end('<html><body><h1>Error 404: ' + fileUrl +

' not a HTML file</h1></body></html>');

}

}

else {

res.statusCode = 404;

}

})

* Start the server, and send various requests to it and see the corresponding responses.
* Do a Git commit with the message "Node HTTP Example 2".

### **Conclusions**

In this exercise you learnt about using the Node HTTP module to implement a HTTP server.

**Introduction to Express**

In this exercise, you will make use of the Express framework to implement similar functionality as implemented by the HTTP module based servers in the previous exercise. At the end of this exercise, you will be able to:

* Implement a simple web server using Express framework
* Implement a web server that serves static content

### **A Simple Server using Express**

* Create a folder named *node-express* in the *NodeJS* folder and move to that folder.
* Copy the *public* folder from *node-http* to this folder.
* At the prompt, type the following to initialize a package.json file in the node-express folder:

npm init

* Accept the standard defaults suggested until you end up with a package.json file containing the following:

{

"name": "node-express",

"version": "1.0.0",

"description": "Node Express Examples",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node index"

},

"author": "Jogesh Muppala",

"license": "ISC"

}

* Then, install the Express framework in the folder by typing the following at the prompt:

npm install express@4.16.3 --save

* Create a file named .gitignore and add the following to it:

node\_modules

* Create a file named *index.js* and add the following code to it:

const express = require('express'),

http = require('http');

const hostname = 'localhost';

const port = 3000;

const app = express();

app.use((req, res, next) => {

console.log(req.headers);

res.statusCode = 200;

res.setHeader('Content-Type', 'text/html');

res.end('<html><body><h1>This is an Express Server</h1></body></html>');

});

const server = http.createServer(app);

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

* Start the server by typing the following at the prompt, and then interact with the server:

npm start

* Initialize a Git repository, add the files and do a Git commit with the message "Express Example".

### **Serving Static Files**

* Install *morgan* by typing the following at the prompt. Morgan is used for logging purposes:

npm install morgan@1.9.0 --save

* Update *index.js* as follows:

. . .

const morgan = require('morgan');

. . .

app.use(morgan('dev'));

app.use(express.static(\_\_dirname + '/public'));

. . .

* Start the server and interact with it and observe the behavior.
* Do a Git commit with the message "Express Serve Static Files".

### **Conclusions**

In this exercise you learnt to use the Express framework to design and implement a web server.

# **Express Router**

### **Objectives and Outcomes**

In this exercise, you will develop a web server that exports a REST API. You will use the Express framework, and the Express router to implement the server. At the end of this exercise, you will be able to:

* Use application routes in the Express framework to support REST API
* Use the Express Router in Express framework to support REST API

### **Setting up a REST API**

* You will continue in the *node-express* folder and modify the server in this exercise.
* Install body-parser by typing the following at the command prompt:

npm install body-parser@1.18.3 --save

* Update *index.js* as shown below:

. . .

* Start the server and interact with it from the browser/postman.
* Do a Git commit with the message "Express Simple REST".

### **Using Express Router**

* Create a new folder named *routes* in the *node-express* folder.
* Create a new file named *dishRouter.js* in the *routes* folder and add the following code to it:

const express = require('express');

const bodyParser = require('body-parser');

const dishRouter = express.Router();

dishRouter.use(bodyParser.json());

dishRouter.route('/')

.all((req,res,next) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

next();

})

.get((req,res,next) => {

res.end('Will send all the dishes to you!');

})

.post((req, res, next) => {

res.end('Will add the dish: ' + req.body.name + ' with details: ' + req.body.description);

})

.put((req, res, next) => {

res.statusCode = 403;

res.end('PUT operation not supported on /dishes');

})

.delete((req, res, next) => {

res.end('Deleting all dishes');

});

module.exports = dishRouter;

* Update *index.js* as follows:

. . .

const dishRouter = require('./routes/dishRouter');

app.use('/dishes', dishRouter);

. . .

* Start the server and interact with it and see the result.
* Do a Git commit with the message "Express Router".

Conclusions

In this exercise, you used the Express framework and Express router to build a server supporting a REST API.