Hyperiondev

TASK

NodeJS

Visit our website

Introduction

WELCOME TO THE NODE.JS TASK!

You have become competent at creating front-end applications at this stage of your learning! Well done! Now it's time to start creating backend applications. In this task, you will learn to use Node.js. You will learn exactly what Node.js is and why it is important to full stack developers. You will also learn how to use some existing Node.js libraries and how to create your own modules using Node.js. By the end of this task, you will be able to use Node.js to create a server object that listens for and responds to HTTP requests.

WHAT IS NODE.JS?

JavaScript was initially designed to run in a browser. Node.js was designed so that JavaScript can be used, not just in a browser on the client-side, but also on web servers. JavaScript is run in a browser using a JavaScript Engine (such as Chrome's V8 engine). *Node.js is an open-source, cross-platform, runtime environment that can run JavaScript files without a browser.* Node.js, therefore, allows us to run JavaScript on a web server! This is great because as web developers we can now use JavaScript to code both the front-end and backend of our web application! Node.js makes full stack web development using JavaScript possible!

Advantages of Using Node.js:

- Great performance: Node.js is good for many common web-development problems as it is designed to optimise throughput and scalability in web applications.
- The code is written in JavaScript: There is no need to learn another programming language. You can write both browser and web server code in JavaScript.

- The node package manager (NPM) provides access to hundreds of thousands of reusable packages: It has best-in-class dependency resolution and can also be used to automate most of the build toolchain.
- It is portable and compatible with most operating systems: It is portable, with versions running on Microsoft Windows, OS X, Linux, Solaris, FreeBSD, OpenBSD, WebOS and NonStop OS. It is also well-supported by many web hosting providers.
- It has a very active developer community: The very active third-party ecosystem and developer community have lots of people who are eager to help.
- Node.js runs single-threaded, asynchronous programming. This means that, even though there is a single call stack that can only handle one instruction at a time, Node.js calls various APIs to handle different instructions. Node.js doesn't wait for API's to complete before going on to the next task. Several tasks can, therefore, run concurrently. Node.js is thus very fast.

Node.js is not the answer for all server-side applications. For example, It is not a good idea to use Node.js for CPU intensive applications.

To write applications or utilities with Node.js, there are two things we first need to do:

- Install Node.js
- Access the modules. As stated previously, JavaScript was originally designed to run in a browser. To be able to run JavaScript on the server, we need some extra functionality (beyond the functionality that we use when we write code to be run in a browser); e.g., we want to be able to use Node.js to access the files system on the computer it is running on. Node.js has a library of built-in modules available for us to use that provide this type of additional functionality. You will use three of the most commonly used modules in this task: the HTTP module, the files system (FS) module and the console module.

INSTALL NODE.JS

Note: *You probably already have Node installed. Check this by typing 'node -v' into your command line interface. If a version number is displayed, Node is already installed!*

Download Node.js and then install it. On Windows and Mac OS X:

1. Go to **https://nodejs.org/** to download the required installer.
2. Download the LTS build that is "Recommended for most users".

3. Click on the downloaded file and follow the installation prompts.

Node is easily installable on Linux machines by using your preferred package manager. E.g.:

```
sudo apt install nodejs
```

ACCESS MODULES

A module is a unit of code that performs a specific task/achieves a specific goal. A module can be a set of functions that you want to use in your code. You can write your own modules that can be used in various pieces of code or you can use libraries of modules that have already been written by other coders.

Node.js runs directly on a computer or server operating system so the environment omits browser-specific JavaScript APIs and adds support for more traditional OS APIs, including HTTP and file system modules.
To include a module, use the **require()** method as shown with the following code:

```
const name_of_variable = require('name_of_module');
```

You will use three of the most commonly used modules in this task: the HTTP module, the files system (FS) module and the console module.

THE HTTP MODULE

Let's get started with our first Node.js program. You will learn the key concepts of working with Node.js as we go.

- **Step 1**: Create a **hello.js** file.
  Since Node.js is JavaScript, you can simply use a text editor (like **Sublime Text** or **Visual Studio**) to create the file.

- **Step 2**: Include required modules.
  The first thing you will do for all your Node.js programs is to include the modules you need for your code to work.

  First, let's include the HTTP module. This module contains all the code needed to use Node.js to transfer data using the HTTP protocol. To see a list

of the methods available with the HTTP module, see **here**. To include the HTTP module, enter the following:

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

- **Step 3**: Create a server object.
  The HTTP module contains a **createServer()** method that can be used to create an HTTP server object. An HTTP server object:
  a. listens for HTTP requests on certain ports on your computer and then
  b. executes a function to handle the request when a request is received.

  This effectively makes your computer a web server.

  The createServer() method takes a callback function as an argument. This function, that will get executed when a request is made, is known as a requestListener. The requestListener takes two objects as arguments:
  - The **request object** that contains properties and methods related to the HTTP request that has been made.
  - The **response object** that contains properties and methods related to the HTTP response that will be sent back to the browser.

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

http.createServer(function(request, response) {
    response.write('Hello World!');
    response.end();
}).listen(3000);
```

  Type the code above. Notice how you are creating a function that will respond to a request event. The write() method is used to write "Hello World!" as a response to the request you receive on port 3000 of your computer. The listen() method is used to specify which port you want to listen for HTTP requests on.

- Step 4: Save your Node.js file. Make sure you save using the .js extension. Take note of the path to this file.

- Step 5: Initiate the Node.js file. Node.js files must be initiated in the command line interface of your computer.
  - Open your command line interface
  - If necessary, change directory so that you are in the same directory as the one in which your Node.js file is stored. e.g: "`cd NodeExamples`"

    - Type: `node hello.js` into the command line (where “hello.js” is the name of the JavaScript file you created in the preceding steps).

- Step 6: Start your browser and navigate to http://localhost:3000/. Voila! You should see your first node.js app in action! To quit the server, you can simply enter CTRL-BREAK (CTRL + C) or exit the command window.

Let’s summarise what you need to do to create a web server using Node.js. You will always have to:

1. Include all required modules.
2. Create a server object.
3. Save your Node.js file and
4. Initiate your Node.js file using the command line

THE FILE SYSTEM MODULE

Another built-in Node.js module is the **file system (fs) module**. This module allows you to manipulate files on your computer. Some of the methods in this module are listed below:

- **fs.open()**
- **fs.readFile()**
- **fs.writeFile()**
- **fs.appendFile()**
- **fs.rename()**
- **fs.unlink()**

All the method names are self-descriptive and it should be clear what each of them does. The possible exception is fs.unlink(). The unlink() method is used to delete files.

To use the file system module you will have to include it as you did with the HTTP module above. e.g. `var fileHandler = require('fs');`

A description of all the available methods in the file system module is available **here**. Make sure you know how to use at least the methods listed above. You will be required to work with some of these in the compulsory task.

It is important to remember that methods are called asynchronously. In other words, Node.js doesn’t wait for one method to finish executing before calling the next method. This could lead to errors. e.g. consider the code below:

```
fs.rename('/tmp/hello', '/tmp/world', (err) => {
  if (err) throw err;
  console.log('renamed complete');
});
fs.stat('/tmp/world', (err, stats) => {
  if (err) throw err;
  console.log(`stats: ${JSON.stringify(stats)}`);
});
```

Since the `rename()` method is called asynchronously, the `stat()` method could complete before the rename method does. To rectify this, use chained callbacks as shown in the following code example:

```
fs.rename('/tmp/hello', '/tmp/world', (err) => {
  if (err) throw err;
  fs.stat('/tmp/world', (err, stats) => {
    if (err) throw err;
    console.log(`stats: ${JSON.stringify(stats)}`);
  });
});
```

CREATING YOUR OWN MODULES

Creating your own modules is easy. Simply create a JavaScript file that contains functions, as you learned to do earlier in this bootcamp, but use the `exports` keyword to make the code available outside the module or JavaScript file.

For example:

- Create a module using the following code:
  ```
  exports.myDateTime = function () {
      return Date();
  };
  ```

- Save the file with the extension **.js** e.g. **myFunction.js**

- Include the module you created where you want to use it as you would any other module, e.g.:
  ```
  const dateTest = require('./testModule.js');
  console.log(dateTest.myDateTime());
  ```

- Save the code above in a file called **test.js**.

- Test the code from the command line by typing `node test` where 'test' is the name of the calling code. Make sure you are in the correct directory.

It is important to understand exactly what the statement `module.exports` does. `module` is an object that represents the current module. `exports` is an object that describes what the current module will make available to calling code. A module can export literals, objects or modules. When you require a module (e.g. `const dateTest = require('./myFunction.js');`)then you get access to the exports object exposed by the module you are requiring. For more information about these objects and how they are used, **see here**. This resource will give you the extra information needed to complete this compulsory task.

A note from our coding mentor

Seraaj

There's a lot more that one can do with Node.js! For example, routing. Instead of learning to do this now, you will first learn to use Express in the next task. This will allow you to write code for server applications much more quickly.

NPM

Besides built-in Node modules and modules that we write ourselves, there are many other libraries of code that you could access. To do this, you can use NPM, a package manager for Node.js. A package is all the files you need in order to use a module. You don't need to install NPM separately because it is automatically installed with Node.js. Before we install and use a package using NPM, let's explore what packages are available for us to use:

- Go to **www.npmjs.com** and create an account with them.
- Once you have created an account and logged in, you should see a number of free packages. Notice that some of the 'Popular Libraries' are tools we will use later like Express and React. You can 'Discover packages' by type, e.g. packages that deal with maths, CSS, robotics etc.
- See what back-end packages are available.

Once you have found a package, it is easy to use it.

- Step 1: Download the package using the CLI. To do this, type: npm install name_of_package, e.g. `npm install chalk`

  To see what the package 'chalk' does, visit **this site**.

- Step 2: Include the package. You do this just as you would include any other module, e.g.
  `const chalk = require('chalk');`

- Step 3: Use the package. www.npmjs.com specifies how the package can be used. Below is an example of code for using 'textics':

```
const chalk = require('chalk');
const log = console.log;

// Combine styled and normal strings
log(chalk.blue('Hello') + ' World' + chalk.red('!'));

//From https://www.npmjs.com/package/chalk
```

Compulsory Task 1

Install Node.js and create a Node.js file called **HelloWorld.js**. This code should create a web server that returns a web page that says “Hey! I can use Node!”. Your web server should listen for HTTP requests on port 3000.

Compulsory Task 2

- Create a module called **palin.js** with a function that finds and returns all the palindromic numbers between 1 and 1000. A palindromic number is one that reads the same forwards and backwards, e.g. 1551; 676; 77

- Create a module called **div7.js** with a function that returns all numbers divisible by 7 between 1 and 100.

- Create a Node.js file called **my_numbers.js** that uses the number modules that you created. The program should:
    - Write the word “Palindromes:” and then the numbers returned by **palin.js** in a text file called **nums.txt**.
    - It should then append the heading “Div7s:” and the numbers returned by **div7.js** to **nums.txt**.
    - Write a message to the console that indicates whether the file was created successfully or not.
    - Read **nums.txt** and display the contents of this file in the browser when the user navigates to **http://localhost:8000/**

- Create a module called **delete_file.js** that will delete **nums.txt** when called.
- Delete the file immediately after displaying its contents from above.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.