Node.js

Getting MEAN 2 – A Practical Workshop

Sharif Malik | 2017

TABLE OF CONTENT

INTRODUCTION

What is NodeJS

- *Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.*
- Node.js was developed by Ryan Dahl in 2009.
- Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications.
- Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.
- *Node.js uses an* ***event-driven***, ***non-blocking I/O*** *model that makes it lightweight and efficient.*

Node.js = Runtime JavaScript + JavaScript Library

Features

1. **Asynchronous and Event Driven :** All APIs of Node.js library are asynchronous, i.e non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.

2. **Super-Fast:** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

3. **Single Threaded** but highly scalable Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.

4. **No Buffering:** Node.js applications never buffer any data. These applications simply output the data in chunks.

5. **License:** Node.js is released under the MIT license.

Who are using

To check who are using Node JS, please go click here.
Recommended: Watch the videos embedded on the page.

Some of the well-known companies are: Netflix, PayPal, Uber, Go Daddy, Microsoft, IBM, Yahoo!, and Yammer, and the list go on...

Where to Use

Node.js is proving itself as a perfect technology partner in these areas:

- JSON APIs based Applications
- Single Page Applications
- I/O bound Applications
- Data Streaming Applications

Where NOT to use

It is not advisable to use Node.js for CPU intensive applications.

Installation

If you want my help for installation, then please click here ;)
You can download the latest version of Node.js installable archive file from Node.js link.

Mac OSX

- Step 1: Open the Terminal app and type `brew update`. This updates Homebrew with a list of the latest version of Node.
  Type `brew install node`.
- Step 2: Sit back and wait, Homebrew has to download some files and install them. But that's it.

Ubuntu OS (Linux Machine)

- Step 1: Open the Terminal app and type `sudo apt-get install nodejs`.
- Step 2: Sit back and wait, Terminal has to download some files and install them. That's all.

Windows OS

- Step 1: Download the Windows installer from the Nodes.js web site.
- Step 2: Run the installer(the .msi file you downloaded in the previous step).
- Step 3: Follow the prompts in the installer (Accept the license agreement, click the NEXT button a bunch of times and accept the default installation settings)
- Step 4: Restart any open command prompt for the change to take effect.

Verify installation:

- Step1: Open terminal and type `node –v` which results some version. e.g. `v8.1.3` on my machine.
- Step 2: Type `npm -v` which results some version. e.g. `5.3.0`

Practical: Lets dive into code

First Application

Let's run the basic JavaScript using Node.js runtime.

Download the example1.js file from provided github link (https://github.com/virtualSharif/gettingMEAN2/blob/master/node/examples/example1.js)

File contains below code:

```
console.log("Hello, Node.js!");
```

Now open the Command Prompt and execute the example1.js using Node.js interpreter to see the result:

i.e. `node example1.js`

If everything is fine with your installation, it should produce the result as follows:

```
Hello, Node.js!
```

Second Application

Explanation:

- Before we dive into creating an actual "Hello, World!" application using Node.js. Let us see the components of a Node.js program and understand one by one.
- A Node.js application consists of the three important components:

1. **Import required modules:** We use the require directive to load Node.js modules.
2. **Create server:** A server which will listen to client's requests like Apache HTTP Server.
3. **Read request and return response:** The server created in an earlier step will read the HTTP request made by the client which can be a browser and return the response.

Creating Application:

- **Code**:

Download the example2.js file from provided github link

(https://github.com/virtualSharif/gettingMEAN2/blob/master/node/examples/example2.js) or find the code below:

```
//Step 1: require module
var http = require("http");

//Step 2: create server
http.createServer(function (request, response){

    //create response head
    response.writeHead(200, {'Content-type': 'text/plain'});

    //send the response Body
    response.end('Hello, Node.js!');

})
//listening to the port
.listen(19991);

//console will print the message
console.log('Server running at http://127.0.0.1:19991');
```

- **Explanation**:

Step 1: Require module
We use the require directive to load the http module and store the returned HTTP instance into an http variable.

Step 2 : Create server
We use the created http instance and call **http.createServer()** method to create a server instance and then we bind it at port 19991 using the listen method associated with the server instance. Pass it a function with parameters request and response. Write the sample implementation to always return "Hello, Node.js!"

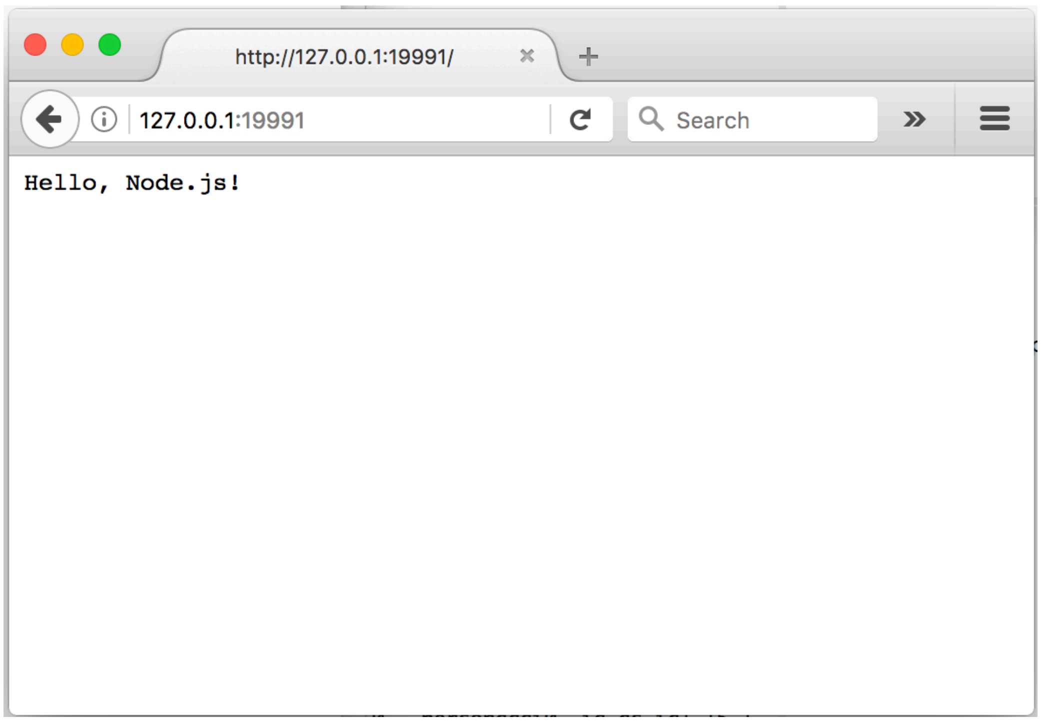
❖ **Verify the Output**:

```
$ node example2.js
```

```
Server running at http://127.0.0.1:19991
```

Make a request to Node.js Server using any browser:

Open http://localhost:19991/ or http://127.0.0.1:19991, and observe the below result.



```
Hello, Node.js!
```

Congratulations! you have your first HTTP server up and running which is responding to all the HTTP requests at port 19991

NPM

Node Package Manager (NPM) provides two main functionalities:

1. Online repositories for node.js packages/modules which are searchable on search.nodejs.org

2. Command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

Note : Now, We don't have to separate install npm. NPM comes bundled with Node.js.

Installing Modules

There is a simple syntax to install any Node.js module through npm:

```
$ npm install <module name>
```

Example: Command to install a famous Node.js web framework module called express is as follows:

```
$ npm install express
```

Now you can use this module in your js file as following:

```
var express = require('express');
```

Global v/s Local Installation:

- By default, NPM installs any dependency in the local mode.
- Local mode refers to the package installation in **node_modules** directory lying in the folder where Node application is present.
- All the installed packages are accessible via **require()** method.

Example:
When we installed express module, it created **node_modules** directory in the current directory which consist express module.

Try this command, and check the current directory for the result:

```
$npm install express
```

After completing the installation of express, In the current directory, **node_modules** directory will be created and under that you can find the express directory.

- Globally installed packages/dependencies are stored in system directory.
- Such dependencies can be used in CLI (Command Line Interface) function of any node.js but cannot be imported using require() in Node application directly.
- Now let's try installing the express module using global installation.

  ```
  $ npm install -g express
  ```

  This will produce similar result but the modules will be installed globally.

Note: you can also use npm ls command to list the local node_ modules.

Package.json

Using package.json - package.json is present in the root directory of any Node application/module and is used to define the properties of a package.

Attributes of package.json :
name - name of the package
version - version of the package
description - description of the package
homepage - homepage of the package
author - author of the package
contributors - name of the contributors to the package
dependencies - list of dependencies. NPM automatically installs all the dependencies mentioned here in the node_module folder of the package.
repository - repository type and URL of the package
main - entry point of the package

keywords – keywords

For example: You can check the package.json under express directory of node_modules.

Uninstalling Modules

Use the following command to uninstall a Node.js module.

```
$npm uninstall express
```

You can also create your own module using npm commands and publish it server for public use. (for more details, please have a look https://www.npmjs.com/)

Callback

- ❖ Callback is an asynchronous equivalent for a function.
- ❖ Callback function is called at the completion of a given task.
- ❖ Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

Example:
Consider a function whose task is read a file content. This function starts reading a file asynchronously and return the control to the execution environment immediately. Hence, next instruction can be executed.
Once file I/O is complete, it will call the callback function with the file content as a input parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Blocking Code Example

❖ Code:

You can download the source code from here(https://github.com/virtualSharif/gettingMEAN2/blob/master/node/examples/example3.js)

OR

a. Create a text file named input.txt with the following content:

```
This is Getting MEAN 2- A practical workshop.
Any fool can know, the point is to understand!!!!!
```

b. Create a js file named example3.js with the following code:

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');
console.log(data.toString());
console.log("Program Ended");
```

❖ Verify the output:
Now run the example3.js to see the result:

```
sharif@MacBook-Pro examples $ node example3.js
This is Getting MEAN 2- A practical workshop.
Any fool can know, the point is to understand!!!!!
Program Ended
```

Non-Blocking Code Example

❖ Code:

You can download the source code from here (https://github.com/virtualSharif/gettingMEAN2/blob/master/node/examples/example4.js)

OR

Create a text file named input.txt with the following content:

```
This is Getting MEAN 2- A practical workshop.
Any fool can know, the point is to understand!!!!!
```

Create a js file named example4.js with the following code:

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
      if (err)
         return console.error(err);
      console.log(data.toString());
});
console.log("Program Ended");
```

❖ Verify the output:
Now run the example4.js to see the result:

```
sharif@MacBook-Pro examples $ node example4.js
Program Ended
This is Getting MEAN 2- A practical workshop.
Any fool can know, the point is to understand!!!!!
```

Explanation:

The above two examples explain the concept of blocking and non- blocking calls.
- The first example shows that the **program blocks until it reads the file** and then only it proceeds to end the program.

- The second example shows that the **program does not wait for file reading** and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the 5le.

Thus, a blocking program executes very much in sequence. From the programming point of view, it is easier to implement the logic but non-blocking programs do not execute in sequence.

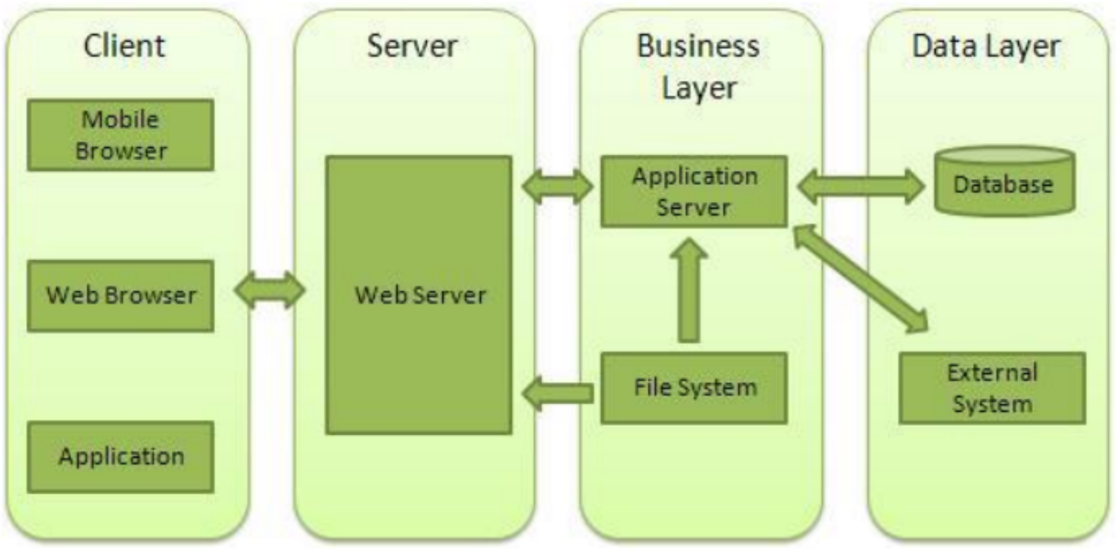
In case a program needs to use any data to be processed, it should be kept within the same block to make it sequential execution.

Web Module

- ❖ A Web Server is a software application which handles HTTP requests sent by the HTTP client, like web browsers, and returns web pages in response to the clients.
- ❖ Web servers usually deliver html documents along with images, style sheets, and scripts.
- ❖ Most of the web servers support server-side scripts, using scripting languages or redirecting the task to an application server which retrieves data from a database and performs complex logic and then sends a result to the HTTP client through the Web server.
- ❖ Apache web server is one of the most commonly used web servers. It is an open source project.

Web Application Architecture

A Web application is usually divided into four layers:



Client - This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.
Server - This layer has the Web server which can intercept the requests made by the clients and pass them the response.
Business - This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.
Data - This layer contains the databases or any other source of data.

Application

Creating a Web Server using Node:

Node.js provides an http module which can be used to create an HTTP server.
Following is the bare minimum structure of the HTTP server which listens at 19991 port.

❖ Code:
You can download the source code from here (https://github.com/virtualSharif/gettingMEAN2/blob/master/node/examples/example5.js) OR

Create file name as example5.js

```
var http = require('http');
var fs = require('fs');
var url = require('url');
//Create the Server
http.createServer(function (request, response){
    //Parse the request containing filename
    var pathname = url.parse(request.url).pathname;
    // Print the name of the file for which request is made.
    console.log("request for "+ pathname + " received");
    // Read the requested file content from file system
    fs.readFile(pathname.substr(1), function (err, data) {
        if(err){
            console.log(err);
            // HTTP Status: 404 : NOT FOUND
            response.writeHead(404, {'Content-Type': 'text/html'});
        } else {
            //HTTP status : 200 : OK
            response.writeHead(200, {'Content-Type': 'text/html'});
            response.write(data.toString());
        }
        //send response body
        response.end();
    });
}).listen(19991);
console.log("Server is running at http://localhost:19991");

```

Create html file named as index.html

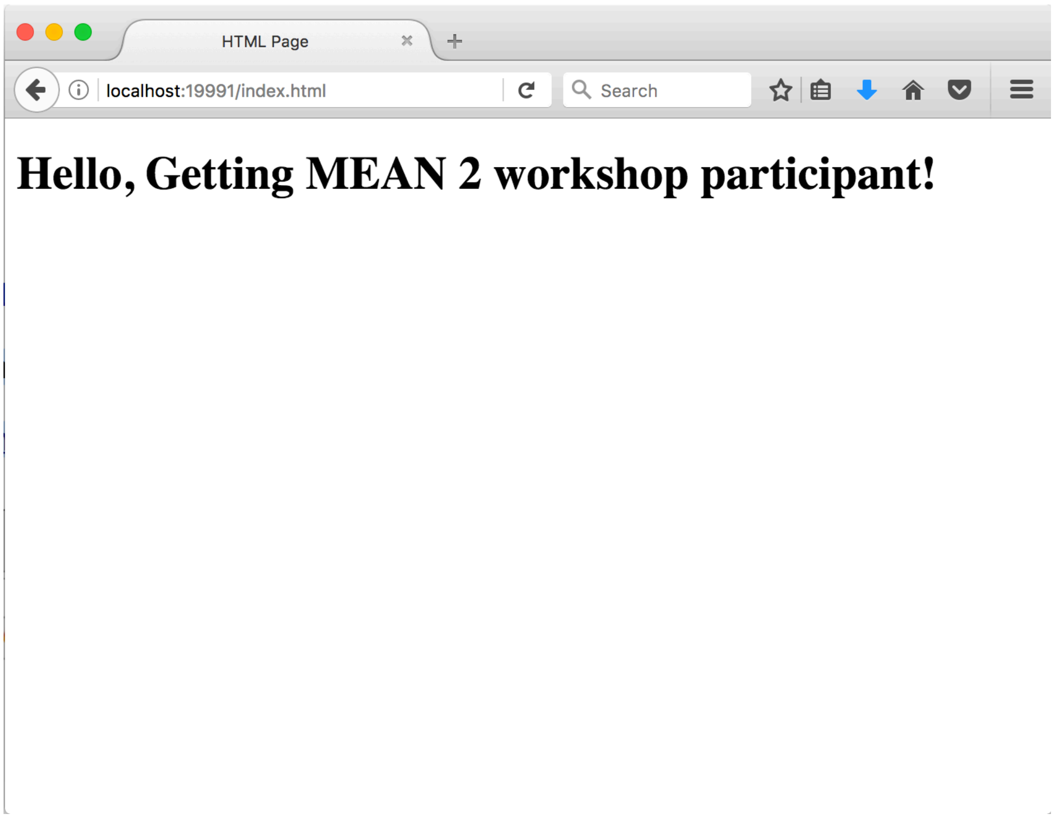
```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML Page</title>
    </head>
    <body>
        <h1>Hello, Getting MEAN 2 workshop participant!</h1>
    </body>
</html>
```

❖ Verify the output:

```
sharif@MacBook-Pro examples $ node example5.js
Server is running at http://localhost:19991
```

Make a request to server by using any browser:

Open **http://localhost:19991/index.html** in any browser to see the following result.



Hello, Getting MEAN 2 workshop participant!

Congratulations! You have created server which serves the backend pages.

References

https://nodejs.org/en/

https://nodeschool.io/