Node.js

Node.js is an asynchronous event driven server-side Javascript runtime environment. It was developed by Ryan Dahl in 2009. Node was designed for building scalable network dependent applications. A rich library of JavaScript modules are also provided for this platform that simplifies the development of web applications.

Features:

Asynchronus

Node.js is asynchronous such that when multiple requests come in it handles them concurrently. Moreover, when a request has been made, Node will move on to the next request even when the previous has not been finished yet.

Event Driven

Node is event driven because as soon as its server starts running, it initiates all variables, declares functions and waits for an event to occur. Node.js’ Event Loop listens for events and triggers a callback function.

Single Threaded but asynchronous

\*\*\*\*\*\*\*\* may explanation pa here

Where to Use Node.js

* I/O bound applications
* Data Intensive Real-Time Applications
* JSON APIs based applications
* Data Streaming applications
* Single Page Applications

Environment Setup

If you are willing to set up your development environment for Node.js then you need a text editor, to develop your own Node.js applications, and the Node.js binary installable.

The Node.js installable can be downloaded from the Official website of Node.js

<https://nodejs.org/en/>

Run the msi file

 - <img src=”version8msi.PNG” slt=”Version 8 installer”>

or

 - <img src=”version10msi.PNG” slt=”Version 10 installer”>

Run the msi files and it will automatically include the Node.js to your machine’s PATH environment variable.

Components of a Node.js application

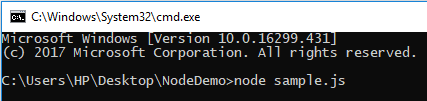
An application developed with Node.js consists of 3 important components:

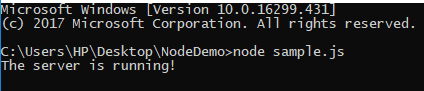
1. Modules – modules are imported for use in your source code for easier and faster coding. Modules are loaded by using the require directive.
2. Server – the server will be the one listening to the requests that clients will make.
3. Request and Response mechanism – when the server listens to a client’s request it generates a response as defined by the programmer in reply to the request.

Sample Code

<img src=“sampleCode.PNG” alt=”Sample Code”>

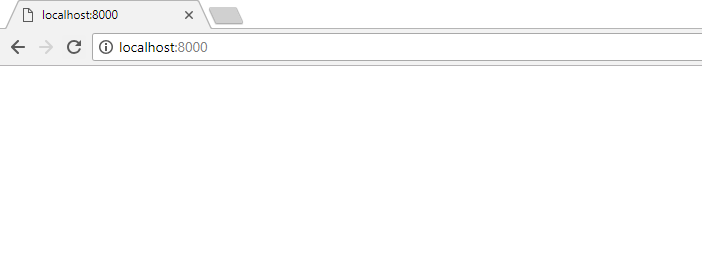
To start the server, go to the directory of your file then run your terminal there. Issue the command :

 - <img src=”sample.PNG” alt=”Run Node.js application command”>

<img src=”” alt=”serverisrunning.PNG”>

Making a Request to the Server

In any browser, open localhost:8000 and observe following result:

<img src=”website.PNG” alt=”hosting url on website”>  - <img src=”samplerun.PNG” alt=”Sample Run”>

Node.js – NPM

The Node Package Manager (NPM) provides 2 main functionalities:

* Node.js packages and modules available via online repositories
* Utility to install packages using a machines command line, also it can do version management and Node.js packages’ mdependency management.

Installing a Module using NPM

To install a module in a command line interface issue the command

 - <img src=”NPMModuleInstallation.PNG” alt=”Install module command”>

Example of a web framework module is the express module.

 - <img src=”NPMExpressModule.PNG” alt=”Express Module Install”>

After installation you can already use the module by including this to your .js file:

- <img src=”NPMExpressModuleUse.PNG” alt=”require(‘express’)”>

Uninstalling a module

To uninstall a Module just issue the following command:

-<img src=”NPMUninstallModule.PNG” alt=”Uninstall module”>

To verify if you have successfully uninstalled a module you can issue the command:

-<img src=”NPMVerifyUninstallation” alt=”Verify Uninstallation”>

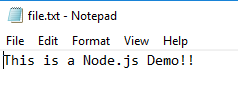
Node.js Callback Concept

A callback is a function equivalent, only synchronous. When a task is completed, a callback function is called. Node.js makes extensive use of callback functions. All Node.js APIs support callbacks.

For example, a function that is programmed to read the contents of a file and return the execution control to the environment immediately in order for the proceeding instructions can be executed. So no code blocking will happen, once the file I/O completes, the callback function is called while it is being passed. This is why Node.js is highly scalable, it can process multiple requests without waiting for a function to return the results.

Blocking Code Example

To simulate the situation above, first, create a text file named file.txt:

- <img src=”fileTXT.PNG” alt=”File Creation”>

Then create a js file named demo.js:

-<img src=”BlockingCodeDemo.png” alt=”Blocking Code”>

Then run the js file that you have created and verify the output:

<img src=”BlockingCodeDemoRun.PNG” alt=”Demonstration”

Non-Blocking Code Example

<img src=”NonBockingCodeExampleDemo.PNG” alt=”Non-Blocking code”>

And run the js file and verify the output: