YASH TRAINING & DEVELOPMENT

TOPIC: Hands on Training for Node JS

INDEX

Node JS

Introduction of Node-JS:-

- Created by Ryan Dahl in 2009
- Development & maintenance sponsored by Joyent
- Last release : 0.10.31
- Based on Google V8 Engine
- +99 000 packages
- Its asynchronous,Non-Blocking, Event-driven server side scripting language built on chrome's V8 engine.

How to Install Node JS:

Go to nodejs.org and install latest LTS version of node then
Type command on cmd: node –v

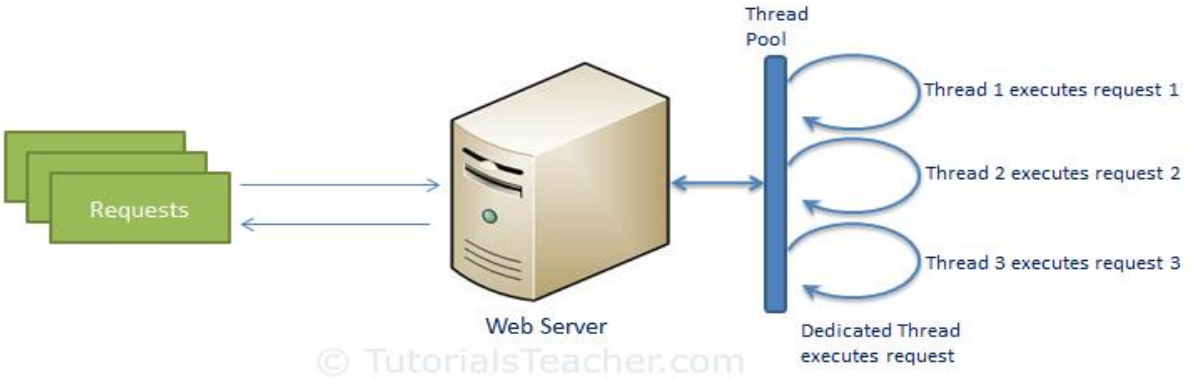
Refer below link for more details:-

https://nodejs.org

Architecture

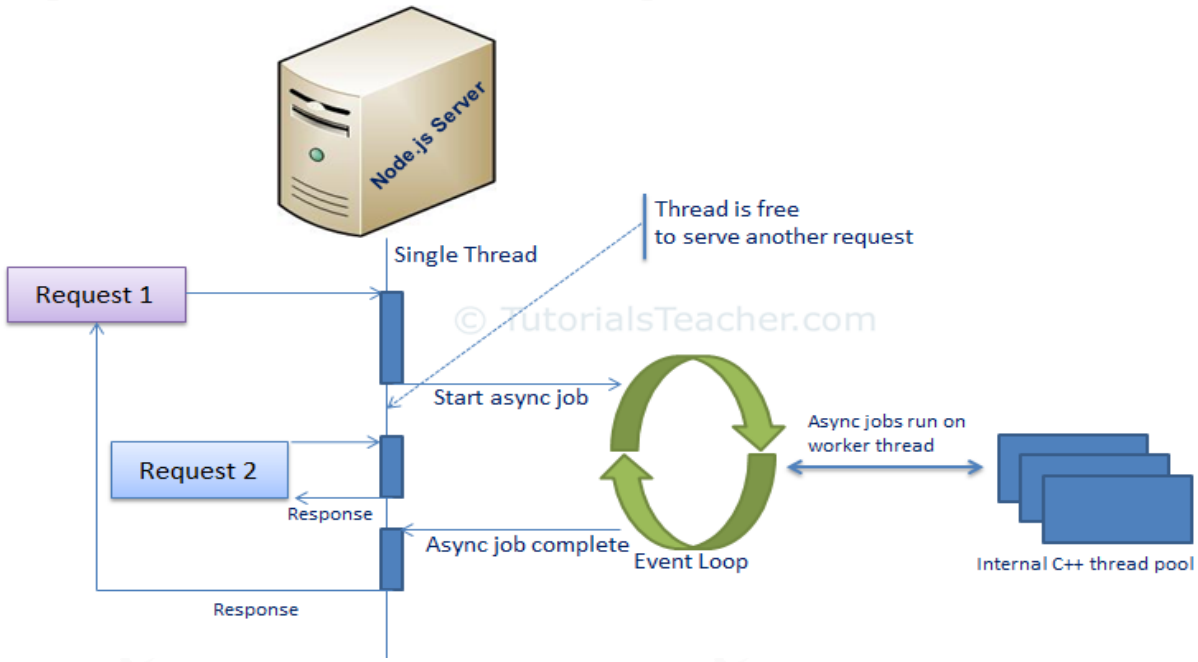
Traditional Web model

`



- In the Traditional Web Model approach all request are accepted by dedicated thread
- It works in synchronized environment
- If no thread is available in the thread pool at any point of time then the request waits till the next available thread.
- Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.

Node JS web Model



- Node JS accepts different request and process it in single thread process unlike traditional model
- All the user requests are handled asynchronously by single thread and doesn't have to wait for the request to complete and is free to handle the next request.
- An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes.
- It uses libev for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

Why Node Js

- Non-Blocking I/O
- V8 JavaScript Engine
- Single Thread with Event Loop
- 40,025 modules
- Windows, Linux, Mac
- 1 Language for Frontend and Backend

- Active community

Features of Node JS

- Asynchronous i/o framework
- A platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications
- Can handle thousands of Concurrent connections with Minimal overhead (cpu/memory) on a single process
- SSJS -> Server-Side JavaScript
- Executed by V8 JavaScript engine the same used by Google chrome browser.
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

REPL for Node

- REPL represents read eval print loop
- Read: It reads user's input; parse the input into JavaScript data-structure and stores in memory.
- Eval: It takes and evaluates the data structure.
- Print: It prints the result.
- Loop: It loops the above command until user press ctrl-c twice.
- Type node on command prompt to get REPL.

Basics of using REPL

`

Enter the program code on cmd and type node to get REPL mode and perform following scripts

```
C:\Users\snehal.pawar>node
> "hello welcome to Node JS"
'hello welcome to Node JS'
> var name="Yash";
undefined
> name
'Yash'
> "Hello"
'Hello'
> 12+12;
24
> console.log(name);
Yash
undefined
```

Array creation with methods

```
> var cars=["Honda","BMW","Hyundai","Renault"];
undefined
> cars;
[ 'Honda', 'BMW', 'Hyundai', 'Renault' ]
> cars.push("Suzuki");
5
> cars;
[ 'Honda', 'BMW', 'Hyundai', 'Renault', 'Suzuki' ]
> cars.pop();
'Suzuki'
> cars;
[ 'Honda', 'BMW', 'Hyundai', 'Renault' ]
>
```

Object creation

```
> var emp={id:1,name:"allen"};
undefined
> emp;
{ id: 1, name: 'allen' }
> emp.name;
'allen'
>
```

Function creation

```
> function add(x,y)
... {
... return x+y;
... }
undefined
> add(2,2);
4
```

Anonymous function creation

```
> var add=function(a,b)
... {
... return a*b;
... }
undefined
> console.log(add(2,3));
6
undefined
>
```

Node Package Manager(NPM)

- NPM is cli tool used for installing, uninstalling, updating packages in your application
- A package in Node.js contains all the files you need for a module.
- Modules are JavaScript libraries you can include in your project.

NPM commands

Install a package/dependencies

```
C:\Users\snehal.pawar>npm install express
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\snehal.pawar\package.json'
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\snehal.pawar\package.json'
npm WARN snehal.pawar No description
npm WARN snehal.pawar No repository field.
npm WARN snehal.pawar No README data
npm WARN snehal.pawar No license field.

+ express@4.17.1
added 45 packages from 32 contributors and audited 704 packages in 2.817s
found 0 vulnerabilities
```

Update the package and check version

```
C:\Users\snehal.pawar>npm update express

C:\Users\snehal.pawar>npm express -v
6.6.0
```

Uninstall package/dependencies

```
C:\Users\snehal.pawar>npm uninstall express
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\snehal.pawar\package.json'
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\snehal.pawar\package.json'
npm WARN snehal.pawar No description
npm WARN snehal.pawar No repository field.
npm WARN snehal.pawar No README data
npm WARN snehal.pawar No license field.

removed 1 package and audited 730 packages in 2.113s
found 0 vulnerabilities
```

Get the list of dependencies in your application

```
C:\Users\snehal.pawar>npm ls
C:\Users\snehal.pawar
+-- accepts@1.3.7
| +-- mime-types@2.1.24
| | `-- mime-db@1.40.0
| `-- negotiator@0.6.2
+-- array-flatten@1.1.1
+-- body-parser@1.19.0
| +-- bytes@3.1.0
| +-- content-type@1.0.4
| +-- debug@2.6.9
| | `-- ms@2.0.0
| +-- depd@1.1.2
| +-- http-errors@1.7.2
| | +-- depd@1.1.2 deduped
| | +-- inherits@2.0.3 extraneous
| | +-- setprototypeof@1.1.1
| | +-- statuses@1.5.0 deduped
| | `-- toidentifier@1.0.0
| +-- iconv-lite@0.4.24
| | `-- safer-buffer@2.1.2
```

- In the above commands we are installing dependencies locally.
- Installing it local, means the module will be available only for a project you installed it (the directory you were in, when ran npm install).
- Global install, instead puts the module into your Node.js path (OS dependent), and will be accessible from any project, without the need to install it separately for each.
- Install dependencies globally

```
C:\Users\snehal.pawar>npm install mysql -g --save
+ mysql@2.17.1
updated 3 packages in 1.12s
```

Search dependencies

```
C:\Users\snehal.pawar>npm search mysql
NAME                        | DESCRIPTION           | AUTHOR           | DATE       | VERSION | KEYWORDS
mysql                       | A node.js driver…     | =dougwilson…     | 2019-04-18 | 2.17.1  |
sqlstring                   | Simple SQL escape…    | =sidorares…      | 2018-02-25 | 2.3.1   | sqlstring sql escape sql escape
typeorm                     | Data-Mapper ORM for…  | =alexmesser…     | 2019-06-04 | 0.2.18  |
sails-mysql                 | MySQL adapter for…    | =balderdashy…    | 2018-11-08 | 1.0.1   | mysql orm waterline sails
sequelize-typescript        | Decorators and some…  | =robinbuschmann  | 2019-08-21 | 1.0.0   | orm object relational mapper sequelize typescript decorators mysql sqlite postg
waterline                   | An ORM for Node.js…   | =balderdashy…    | 2018-11-26 | 0.13.6  | mvc orm mysql postgresql redis mongodb active-record waterline sails sails.js
mysqldump                   | Create a DUMP from…   | =bradzacher…     | 2019-07-24 | 3.1.0   | backup mysql mysql-dump dump restore database backup database
postgrator                  | A SQL migration…      | =damianberesfo…  | 2019-06-05 | 3.10.2  | migrator migration postgres postgresql mysql sql server sql
hapi-plugin-mysql           | Hapi plugin for…      | =adrivanhoudt    | 2019-06-29 | 6.0.2   | hapi mysql plugin
feathers-sequelize          | A service adapter…    | =daffl =ekryski  | 2019-07-22 | 6.0.1   | feathers feathers-plugin sequel sequelize mysql sqlite mariadb postgres pg mssq
think-model-mysql           | Mysql adapter for…    | =berwin =bezos…  | 2019-05-09 | 1.1.0   | thinkjs orm mysql adapter model
loopback-connector-mysql    | MySQL connector for…  | =b-admike…       | 2019-07-25 | 5.4.2   |
seneca-mysql-store          | MySQL database…       | =darsee…         | 2016-08-27 | 1.1.0   | seneca mysql plugin
tunnel-ssh                  | Easy extendable SSH…  | =agebrock        | 2018-02-27 | 4.1.4   | tunnel ssh mysql develop net
marv-mysql-driver           | A mysql marv driver…  | =cressie176      | 2019-08-17 | 2.0.2   | marv database db migration migrate mysql
gatsby-source-mysql         | Source plugin for…    | =malcolmkee      | 2019-08-27 | 2.2.1   | gatsby gatsby-plugin mysql
think-mysql                 | think-mysql for…      | =lushijie…       | 2019-06-03 | 1.4.3   |
sails-mysql-pillow          | Fork of sails-mysql…  | =polochon        | 2016-02-26 | 1.0.1   | mysql orm waterline sails
bs-mysql2                   | ReasonML bindings…    | =scull7          | 2019-07-28 | 9.0.3   | BuckleScript reason reasonml mysql databases node
@tadashi/mysql-pool         | MySQL Pooling…        | =lagden          | 2019-08-24 | 1.3.2   | mysql pool connection query
```

Check Package.json file

```
C:\Users\snehal.pawar>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (snehal.pawar)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\snehal.pawar\package.json:

{
  "name": "snehal.pawar",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "create-react-app": "^3.1.1"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes) y
```

NVM commands

- nvm stands for Node Version Manager. As the name suggests, it helps you manage and switch between different Node versions with ease.
- It provides a command line interface where you can install different versions with a single command, set a default, switch between them and much more.

• First, make sure you uninstall any Node.js version you might have on your system, as they can collide with the installation.
• After this, download the latest stable installer. Run the executable installer, follow the steps provided and you're good to go!
• Install NVM by downloading from "**nvm-setup.zip**" : https://github.com/coreybutler/nvm-windows/releases
• Copy the settings.txt from the "**C:\Users\YOUR_USERNAME\AppData\Roaming\nvm**" to "**C:\Users\ YOUR_USERNAME**"
• Restart your system for the changes to reflect.
• Open the console and type below command :
• nvm list

Install specific version

```
npm install 10.15.3
```

get the list of versions available

```
nvm ls available
```

get the latest version of Node.js as your default

```
nvm alias default 10.15.3
```

To designate the latest version of Node.js as your default

```
nvm alias my-favorite 10.15.3
```

Switch to the latest version of Node.js, you'd simply run

```
nvm use my-favourite
```

check which version you're currently using

```
nvm current
```


`

Modules and its types

- Modules are set of JS libraries/set of functions that we need to include in application.
- Each module can be placed in a separate .js file under a separate folder.
- Node.js includes three types of modules:
- Core Modules
- Local Modules
- Third Party Modules

Creating first node JS program
Create FirstDemo.js file like below

```
console.log("hello welcome to node Js");
var name="Yash";
console.log("Name is : "+name);
```

Run below file on cmd by below command

```
D:\Node-js>set path="C:\Program Files\nodejs";

D:\Node-js>node FirstDemo.js
hello welcome to node Js
Name is : Yash
```

Core Modules

- Node.js is a light weight framework.
- The core modules include bare minimum functionalities of Node.js.
- This are inbuilt modules available in node modules
- We need to import them by using require() method
- var module = require('module_name');

`

- Different core modules available are
- http,util,path,url,querystring,assert,fs,assert,os,etc

OS module

```
const os=require('os');
console.log("os.freemem(): \n",os.freemem());
console.log("os.homedir(): \n",os.homedir());
console.log("os.hostname(): \n",os.hostname());
console.log("os.loadavg(): \n",os.loadavg());
console.log("os.platform(): \n",os.platform());
console.log("os.tmpdir(): \n",os.tmpdir());
console.log("os.totalmem(): \n",os.totalmem());
console.log("os.type(): \n",os.type());
```

Path Module

```
var path = require("path");
console.log(`Full path: ${__filename} and filename is ${path.basenam
// Resolve
console.log('resolve : ' + path.resolve('core-modules.js'));
// Extension
console.log('ext name: ' + path.extname('core-modules.js'));
```

Assert Module

```
var assert = require('assert');
function add (a, b) {
  return a + b;
}
var expected = add(1,2);
assert( expected === 3, 'one plus two is three');
```

QueryString Module

```
var querystring = require('querystring');
var q = querystring.parse('year=2019&month=April');
console.log(q.year);
```

Util module

```
var util = require('util');
var txt = 'Congratulate %s on his %dth birthday!';
var result = util.format(txt,'Allen',1);
console.log(result);
```

Note-Trainees needs to explore other core modules

Local Module

- Local Modules are user created modules
- As the name says it is local in nature we cannot use them anywhere in application.
- So to make it global we used module.exports or exports
- It can be used by require("module");

Create a local module and save with Demo.js

```
module.exports="hello world";
```

Create another file with name app.js to access above local module

```
var a=require("./Demo.js");
console.log(a);
```

Output:

```
D:\Node-js>node App.js
hello world
```

As export is an object we can attach Properties and methods to it.Please write below code in Demo.js

```
exports.myMessage = 'Hello world';
```

In app.js calling local module

```
var a=require("./Demo.js");
console.log(a.myMessage());
```

Output:

```
D:\Node-js>node App.js
{ myMessage: 'Hello world' }
Hello world
```

Attaching function to export object

```
module.exports.show = function (msg) {
    console.log(msg);
};
```

Calling local module in app.js

```
module.exports.show = function (msg) {
    console.log(msg);
};
```

Output:

```
D:\Node-js>node App.js
Hello welcome to show function
```

Attaching object to module.export

```
module.exports={id:1,name:'Allen'}
```

Calling local module in App.js

```
var a=require("./Demo.js");
console.log(a.id+""+a.name);
```

Output:

```
D:\Node-js>node App.js
1Allen
```

Attaching anonymous function to export object

```
module.exports= function (msg) {
    console.log(msg);
}
```

Calling local module in app.js

```
var a=require("./Demo.js");
console.log(a("Hello welcome to anonymous function"));
```

Output:

```
D:\Node-js>node App.js
Hello welcome to anonymous function
undefined
```

Attaching class to export object

```
module.exports = function (firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = function () {
        return this.firstName + ' ' + this.lastName;
    }
}
```

Calling local module in app.js

```
var info = require('./Demo.js');
var myinfo = new info('Allen', 'Turin');
console.log(myinfo.fullName());
```

Output:

```
D:\Node-js>node App.js
Allen Turin
```

Third Party Module

- The third party module can be downloaded by NPM (Node Package Manager).
- These type of modules are developed by others and we can use that in our project.
- Some of the best third party module examples are listed as follows: express, gulp, lodash, async, socket.io, mongoose, underscore, pm2, bower, q, debug, react, mysql,mocha etc.
- Third party modules can be install inside the project folder or globally.
- Local installation of third party module

```
D:\Node-js>npm install express
```

Global installation of third party module

```
D:\Node-js>npm install -g --save express
```

- To load a module in your node application you can just use "require()" function. whose syntax is given below.

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

HTTP Module

- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
- To include the HTTP module, use the require() method:

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

- HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

- Use the createServer() method to create an HTTP server:

```
http.createServer(function (req, res) {
```

- We can add HTTP Header

```
res.writeHead(200, {'Content-Type': 'text/html'});
```

Creating complete HTTP module for running scripts on server
Http_module.js file below

```
var http = require('http');
//create a server object:
http.createServer(function (req, res) {
//creating header
res.writeHead(200, {'Content-Type': 'text/html'});
//write response
res.write('Hello World!'); //write a response to the client
res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
console.log("my server is running on port 8080");
```

Go to cmd and run following command to start the server

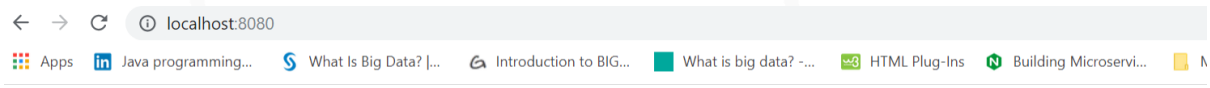
```
D:\Node-js>node Http_Module1.js
my server is running on port 8080
```

Go to browser and run following url

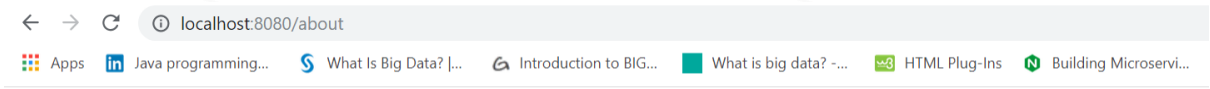
Handling HTTP request

```
var http = require('http'); // Import Node.js core module
 //create web server
var server = http.createServer(function (req, res) {
if (req.url == '/') { //check the URL of the current request
 // set response header
res.writeHead(200, { 'Content-Type': 'text/html' });
// set response content
res.write('<html><body><p>This is home Page.</p></body></html>
res.end();}
else if (req.url == "/about") {
res.writeHead(200, { 'Content-Type': 'text/html' });
res.write('<html><body><p>This is about Page.</p></body></html
res.end();}
else if (req.url == "/admin") {
res.writeHead(200, { 'Content-Type': 'text/html' });
res.write('<html><body><p>This is admin Page.</p></body></html
res.end();}else
res.end('Invalid Request!');
});server.listen(8080); //6 - listen for any incoming requests
console.log('Node.js web server at port 8080 is running..')
```

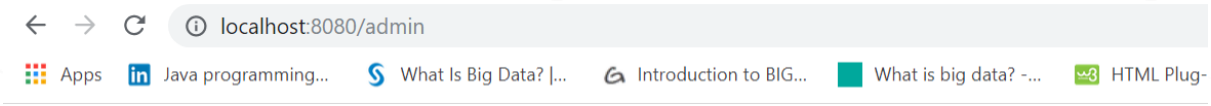
Output:

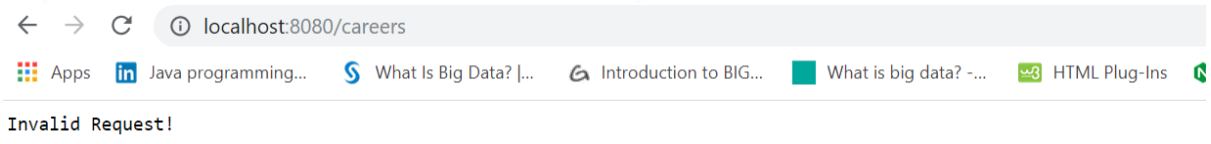


This is home Page.



This is about Page.





@Note-Trainees to explore how to send JSON data on server in HTTP modules

File System

- fs module is one of the core module to access physical file system.
- The fs module is responsible for all the asynchronous or synchronous file I/O operations.
- Different operation used in fs are
- fs.readFile()
- fs.open()
- fs.appendFile()
- fs.writeFile()
- fs.unlink()
- fs.rename()
- uploadFiles

Method	Description
fs.readFile(fileName [,options], callback)	Reads existing file.
fs.writeFile(filename, data[, options], callback)	Writes to the file. If file exists then overwrite the content otherwise creates new file.
fs.open(path, flags[, mode], callback)	Opens file for reading or writing.
fs.rename(oldPath, newPath, callback)	Renames an existing file.
fs.chown(path, uid, gid, callback)	Asynchronous chown.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.link(srcpath, dstpath, callback)	Links file asynchronously.
fs.symlink(destination, path[, type], callback)	Symlink asynchronously.
fs.rmdir(path, callback)	Renames an existing directory.
fs.mkdir(path[, mode], callback)	Creates a new directory.
fs.readdir(path, callback)	Reads the content of the specified directory.
fs.utimes(path, atime, mtime, callback)	Changes the timestamp of the file.
fs.exists(path, callback)	Determines whether the specified file exists or not.
fs.access(path[, mode], callback)	Tests a user's permissions for the specified file.
fs.appendFile(file, data[, options], callback)	Appends new content to the existing file.

Opening a file

```
var fs = require('fs');
fs.open('file2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});
```

Output:

```
D:\Node-js>node Openfile.js
Saved!
```

Reading a file through console. Create one file for reading.

```
var fs = require('fs');
fs.readFile('file1updated.txt', function (err, data)
{ if (err) throw err;
console.log(data.toString());
});
```

Output:

```
D:\Node-js>node Readfile.js
Hello welcome to Fs module for IO operations
```

Asynchronous Reading through console

```
var fs = require("fs");
Asynchronous read
fs.readFile('Data1updated.txt', function (err, data) {
   if (err) {
       return console.error(err);
   }
   console.log("Asynchronous read 1: " + data.toString());
});
fs.readFile('file2.txt', function (err, data) {
   if (err) {
       return console.error(err);
   }
   console.log("Asynchronous read 2: " + data.toString());
});
fs.readFile('Data1updated.txt', function (err, data) {
   if (err) {
       return console.error(err);
   }
   console.log("Asynchronous read 3: " + data.toString());
});
```

Output:

```
D:\Node-js>node Aync-Sync_Read.js
Asynchronous read 1: updated file
Asynchronous read 3: Hello welcome to streams
Asynchronous read 2: Hello welcome to Fs module for IO operations
```

Synchronous reading

```
// Synchronous read
var fs = require("fs");
var data = fs.readFileSync('Data1.txt');
console.log("Synchronous read 1: " + data.toString());
console.log("Program Ended");

// Synchronous read
var data = fs.readFileSync('file1updated.txt');
console.log("Synchronous read 2: " + data.toString());
console.log("Program Ended");

// Synchronous read
var data = fs.readFileSync('Data5.txt');
console.log("Synchronous read 3: " + data.toString());
console.log("Program Ended");
```

Output:

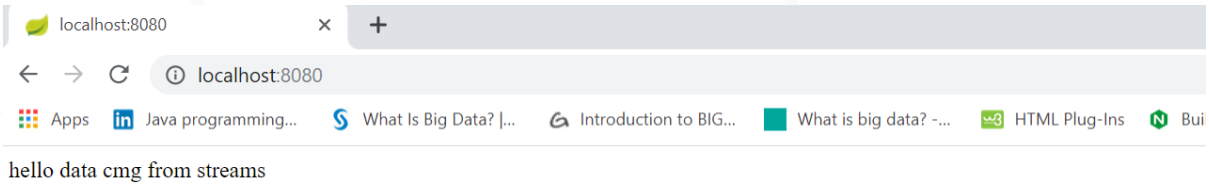
```
D:\Node-js>node Aync-Sync_Read.js
Synchronous read 1: updated file
Program Ended
Synchronous read 2: Hello welcome to Fs module for IO operations
Program Ended
Synchronous read 3: Hello welcome to streams
Program Ended
```

Reading file through web

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('Data1updated.txt', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(8080);
console.log("Server started");
```

Output:

```
D:\Node-js>node Readfile.js
Server started
```



Rename a file

```
var fs=require("fs");
fs.rename("Data1updated.txt","Data1Updated2.txt",function(err,data){
if (err) throw err;
console.log("File is renamed");
});
```

Output:

```
D:\Node-js>node Renamefile.js
File is renamed
```

Writing to a file

```
var fs = require('fs');
fs.writeFile('test1.txt', 'Bye Bye!', function (err) {
if (err)
console.log(err);
else
console.log('Write operation complete.');
});
```

Output:

```
D:\Node-js>node Writefile.js
Write operation complete.
```

Deleting a file

```
var fs = require('fs');
fs.unlink('test1.txt', function () {
console.log('deleted.');
});
```

Output:

```
D:\Node-js>node Deletefile.js
deleted.
```

Appending a file

```
var fs = require('fs');
fs.appendFile('test.txt', 'Hello World!updated', function (err) {
if (err)
console.log(err);
else
console.log('Append operation complete.');
});
```

Output:

```
D:\Node-js>node Appendfile.js
Append operation complete.
```

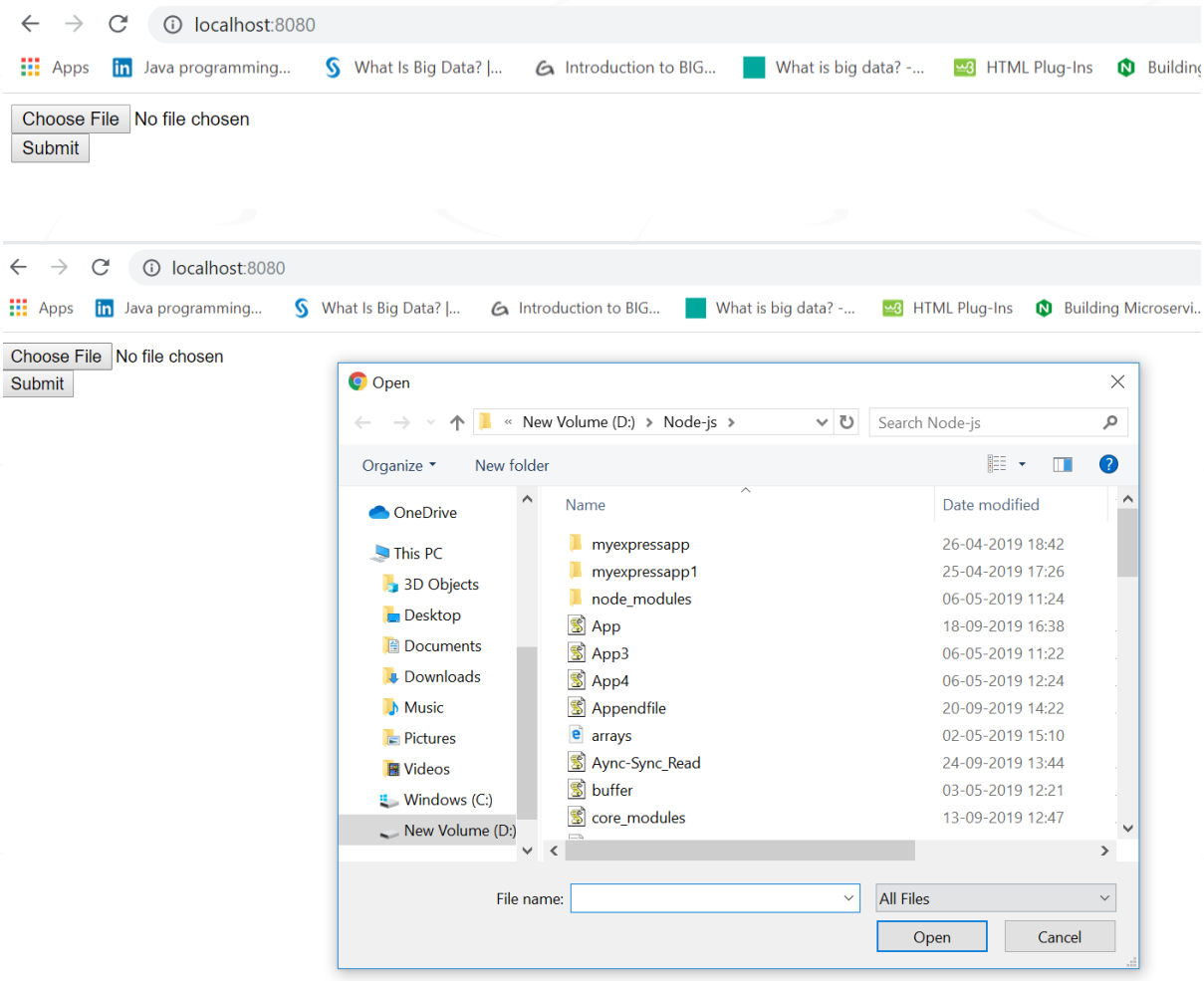
Uploading files
npm install formidable –save

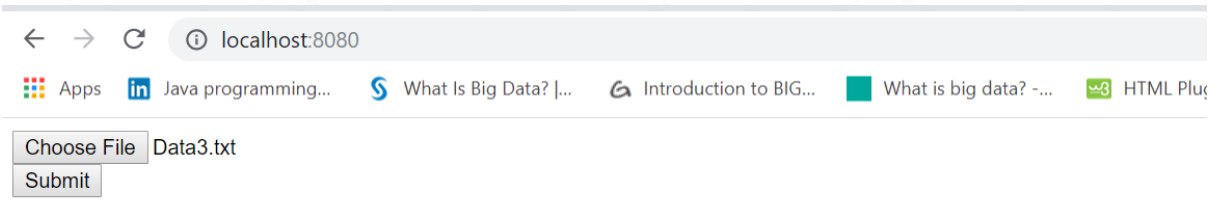
```
var http = require('http');
var formidable = require('formidable');
http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload"
    method="post" enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }).listen(8080);
//try to change the location of uploaded file from
//temporary folder to desired location
```

Go to cmd run above file

```
D:\Node-js>node Uploadfile.js
```

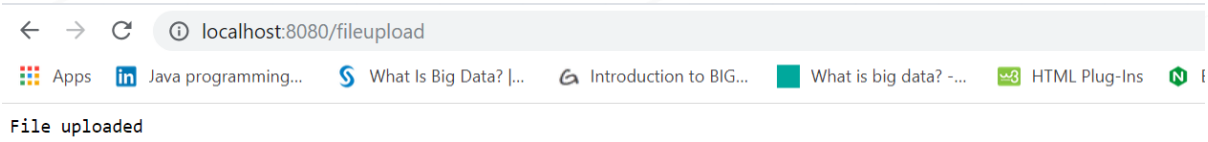
Go to browser





Click on submit button

`



Streams

- Streams are objects that let you read data from a source or write data to a destination in continuous fashion. In Node.js, there are four types of streams −
- Readable − Stream which is used for read operation.
- Writable − Stream which is used for write operation.
- Duplex − Stream which can be used for both read and write operation.
- Transform − A type of duplex stream where the output is computed based on input.
- Each type of Stream is an EventEmitter instance and throws several events at different instance of times.
- data − This event is fired when there is data is available to read.
- end − This event is fired when there is no more data to read.
- error − This event is fired when there is any error receiving or writing data.
- finish − This event is fired when all the data has been flushed to underlying system.

Reading streams

```
var fs = require("fs");
var data = '';
// Create a readable stream
var readerStream = fs.createReadStream('Data1updated.txt');
// Set the encoding to be utf8.
readerStream.setEncoding('UTF8');
// Handle stream events --> data, end, and error
readerStream.on('data', function(chunk) {
   data += chunk;
});
readerStream.on('end',function() {
   console.log(data);
});
```

Output:

```
D:\Node-js>node streams.js
hello data cmg from streams
```

Writing to streams

`

```
var fs = require("fs");
var data = 'Hello welcome to streams';
// Create a writable stream
var writerStream = fs.createWriteStream('Data5.txt');
// Write the data to stream with encoding to be utf8
writerStream.write(data,'UTF8');
// Mark the end of file
writerStream.end();
// Handle stream events --> finish, and error
writerStream.on('finish', function() {
   console.log("Write completed.");
});
writerStream.on('error', function(err) {
   console.log(err.stack);
});
console.log("Program Ended");
```

Output:

```
D:\Node-js>node streams.js
Program Ended
Write completed.
```

Creating readable stream to read from one file and then writing it to another file by creating writable stream

```
var fs = require("fs");
// Create a readable stream
var readerStream = fs.createReadStream('Data5.txt');
console.log("File Read from first file");
// Create a writable stream
var writerStream = fs.createWriteStream('Data1.txt');
// Pipe the read and write operations
// read input.txt and write data to output.txt
readerStream.pipe(writerStream);
console.log("File written to another file");
console.log("Program Ended");
```

Output:

```
D:\Node-js>node streams.js
File Read from first file
File written to another file
Program Ended
```

```
Note:try to work on chaining of streams,compressing and decompressing file
```

Buffers:

- Node.js provides Buffer class to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap.
- Buffer class is used because pure JavaScript is not nice to binary data. So, when dealing with TCP streams or the file system, it's necessary to handle octet streams.
- Buffer class is a global class. It can be accessed in application without importing buffer module
- var buf = new Buffer(10);
- var buf = new Buffer([10, 20, 30, 40, 50]);

`

- var buf = new Buffer("Simply Easy Learning", "utf-8");

Creating and getting length of buffer

```
var buf = new Buffer(20);//represents length
var len = buf.write("Simply Easy Learning");
console.log("The length is"+buf.length);//getting length
console.log("Octets written : "+  len);
console.log(buf);
```

Output:

```
D:\Node-js>node buffer.js
The length is20
Octets written : 20
<Buffer 53 69 6d 70 6c 79 20 45 61 73 79 20 4c 65 61 72 6e 69 6e 67>
(node:13816) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the B
uffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
```

Converting buffer into Json

```
var buf = new Buffer(20);//represents length
var len = buf.write("Simply Easy Learning");
var json = buf.toJSON(buf);//convert buffer into json
console.log(json);
```

Output:

```
D:\Node-js>node buffer.js
{ type: 'Buffer',
  data:
   [ 83,
     105,
     109,
     112,
     108,
     121,
     32,
     69,
     97,
     115,
     121,
     32,
     76,
     101,
     97,
     114,
     110,
     105,
     110,
     103 ] }
(node:25092) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the B
uffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
```

Reading from buffers

```
var buf = new Buffer(26);//reading from buffers
for (var i = 0 ; i < 26 ; i++) {
  buf[i] = i + 97;
}
console.log( buf.toString('ascii'));       // outputs: abcdefghijklm
console.log( buf.toString('ascii',0,5));   // outputs: abcde
console.log( buf.toString('utf8',0,5));    // outputs: abcde
console.log( buf.toString(undefined,0,5)); // encoding defaults to '
```

Output:

```
D:\Node-js>node buffer.js
abcdefghijklmnopqrstuvwxyz
abcde
abcde
abcde
(node:21912) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the B
uffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
```

Concatenation of buffers

```
var buffer1 = new Buffer('Hello All ');//concatenation of buffers
var buffer2 = new Buffer('Welcome to Node Buffer');
var buffer3 = Buffer.concat([buffer1,buffer2]);
console.log("buffer3 content: " + buffer3.toString());
```

Output:

```
D:\Node-js>node buffer.js
buffer3 content: Hello All Welcome to Node Buffer
(node:24944) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the B
uffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
```

Comparing buffers

```
var buffer1 = new Buffer('HELLO');//Comparing Buffers
var buffer2 = new Buffer('hellooooooo');
var result = buffer1.compare(buffer2);
console.log(result);
```

Output:

```
D:\Node-js>node buffer.js
-1
(node:15824) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(), Buffer.allocUnsafe(), or Buffer.from(
) methods instead.
```

Copying buffers

```
var buffer1 = new Buffer('Hello All');
var buffer2 = new Buffer(12);
buffer1.copy(buffer2);//copy a buffer
console.log("buffer2 content: " + buffer2.toString());
```

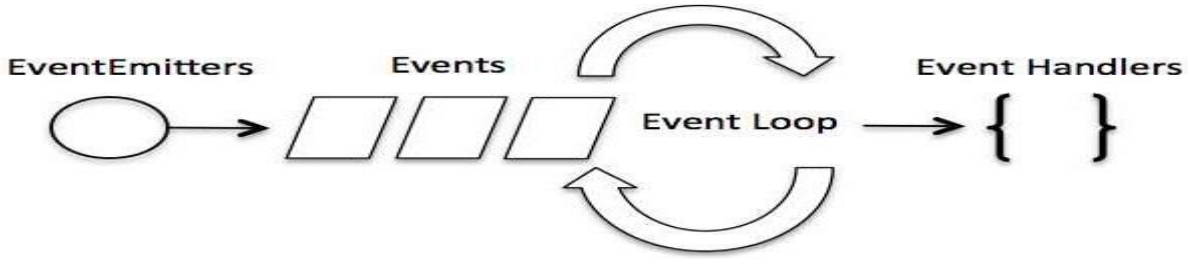
Output:

```
D:\Node-js>node buffer.js
buffer2 content: Hello All
(node:24612) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(), Buffer.allocUnsafe(), or Buffer.from(
) methods instead.
```

Note: Trainees need to explore slice() and other methods of buffers

Events:

- Node js has built-in module to create and fire events
- It uses events heavily and initiates variables, function and waits for the events to execute.
- Node JS being single threaded supports concurrency via event and callbacks
- Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.
- Node js uses event driven programming so it is faster because it initiates variables and declares functions and then simply waits for the event to occur.
- In event-driven application, there is main loop that listens for events, and then triggers a callback function when one of those events is detected.



Steps for events

Import events module

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

Create an eventEmitter object

```
var eventEmitter = new events.EventEmitter();
```

Following is the syntax to bind an event handler with an event −
Bind event and event handler as follows

```
eventEmitter.on('eventName', eventHandler);
```

We can fire an event programmatically as follows −

Fire an event

eventEmitter.emit('eventName');

Methods of Eventemitter class:

• emitter.on(event,listener)
• emitter.emit(emit,args[],args[],args[])
• emitter.addlistener(event,listener)
• emitter.removelistener(event,listener) etc.
creating first event demo

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
//Create an event handler:
var myEventHandler = function () {
console.log('This is my first event!');
}
//Assign the event handler to an event:
eventEmitter.on('firstevent', myEventHandler);
eventEmitter.emit('firstevent');
```

Output:

```
D:\Node-js>node Event1.js
This is my first event!
```

Creating event along with method

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
//Create an event handler:
var myEventHandler = function () {
console.log('This is my first event!');
}
//Assign the event handler to an event:
eventEmitter.on('firstevent', myEventHandler);
//eventEmitter.emit('firstevent');
function show()
{
var a=12;
console.log(a);
}
//Fire the 'firstevent' event: along with function
eventEmitter.emit('firstevent',show());
```

Output:

```
D:\Node-js>node Event1.js
12
This is my first event!
```

Creating and emitting multiple events

```
var emitter = require('events').EventEmitter;
var em = new emitter();
//Subscribe FirstEvent
em.addListener('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});
//Subscribe SecondEvent
em.on('SecondEvent', function (data) {
    console.log('Second subscriber: ' + data);
});
// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter exampl
// Raising SecondEvent
em.emit('SecondEvent', 'This is my second Node.js event emitter exam
```

Output:

```
D:\Node-js>node Events2.js
First subscriber: This is my first Node.js event emitter example.
Second subscriber: This is my second Node.js event emitter example.
```

Emitting events within events

```
var events = require('events'); //Import events module
var eventEmitter = new events.EventEmitter();// Create an eventEmitt
var connectHandler = function connected() {  // Create an event hand
console.log('connection succesful.');
eventEmitter.emit('data_received'); // Fire the data_received event
eventEmitter.emit('new_event');   }
eventEmitter.on('connection', connectHandler);// Bind the connection
// Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function(){
console.log('data received succesfully.');});});
eventEmitter.on('new_event', function(){
var name="Allen";
console.log(name+""+name2);
var name2="John";
var cars=["Honda","BMW"];
console.log('I am in new event');
});
// Fire the connection event
eventEmitter.emit('connection');
console.log("Program Ended.");
```

Output:

```
D:\Node-js>node Event3.js
connection succesful.
data received succesfully.
Allenundefined
I am in new event
Program Ended.
```

Timers:

- Node.js Timer functions are global functions.
- don't need to use require() function in order to use timer functions. Let's see the list of timer functions.
- Set timer functions:
- setImmediate(): It is used to execute setImmediate.

- setInterval(): It is used to define a time interval.
- setTimeout(): ()- It is used to execute a one-time callback after delay milliseconds.
- Clear timer functions:
- clearImmediate(immediateObject): It is used to stop an immediateObject, as created by setImmediate
- clearInterval(intervalObject): It is used to stop an intervalObject, as created by setInterval
- clearTimeout(timeoutObject): It prevents a timeoutObject, as created by setTimeout

using setInterval

```
var i=1;
setInterval(function() {
console.log("setInterval: Hey! 1 millisecond completed!..");
//will display statement after every 1 milisec
i=i+10;
console.log(i);
}, 1000);
```

Output:

```
D:\Node-js>node timer.js
setInterval: Hey! 1 millisecond completed!..
11
setInterval: Hey! 1 millisecond completed!..
21
setInterval: Hey! 1 millisecond completed!..
31
setInterval: Hey! 1 millisecond completed!..
41
setInterval: Hey! 1 millisecond completed!..
51
setInterval: Hey! 1 millisecond completed!..
61
^C
```

Using setTimeout

```
setTimeout(function() {
console.log("setTimeout: Hey! 1000 millisecond completed!..");
}, 1000);
```

Output:

```
D:\Node-js>node timer.js
setTimeout: Hey! 1000 millisecond completed!..
```

```
var i=1;
setInterval(function() {
console.log("setInterval: Hey! 1 millisecond completed!..");
//will display statement after every 1 milisec
i=i+10;
console.log(i);
}, 1000);
setTimeout(function() {
console.log("setTimeout: Hey! 1000 millisecond completed!..");
}, 1000);
```

Output:

```
D:\Node-js>node timer.js
setTimeout: Hey! 1000 millisecond completed!..

D:\Node-js>node timer.js
setInterval: Hey! 1 millisecond completed!..
11
setTimeout: Hey! 1000 millisecond completed!..
setInterval: Hey! 1 millisecond completed!..
21
setInterval: Hey! 1 millisecond completed!..
31
setInterval: Hey! 1 millisecond completed!..
41
setInterval: Hey! 1 millisecond completed!..
51
^C
```

Note:Participants needs to explore clear timer functions

Database Connectivity:

- Node.js can be used in database applications.
- One of the most popular databases is MySQL.
- NoSQL databases like MongoDb are the best fit with Node.js.
- To access a MySQL database with Node.js, you need a MySQL driver.
- npm install mysql
- Node.js can use this module to manipulate the MySQL database:
- var mysql = require('mysql');

Steps for Database Connection:

- Create a connection
- Create a database
- Create a table
- Insert record as one or many
- Update record
- Delete record

Creating a connection and checking the connection

`

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "root",
database: "test"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");});});
```

Output:

```
D:\Node-js>node Database-connect.js
Connected!
```

Creating a database

`

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "root",
database: "test"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
var sql = "create database dummy_table";
con.query(sql, function (err, result)
{
if (err) throw err;
console.log("datatable created");
});
});
```

Output:

```
D:\Node-js>node Database-connect.js
Connected!
datatable created
```

Creating a table

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "root",
database: "test"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
var sql = "CREATE TABLE employees1 (id INT, name VARCHAR(255), age I
con.query(sql, function (err, result)
{
if (err) throw err;
console.log("Table created");
});
});
```

Output:

```
D:\Node-js>node Database-connect.js
Connected!
Table created
```

Inserting record into tables

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "root",
database: "test"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
var sql = "INSERT INTO employees2 (id, name, age, city) VALUES ('1',
con.query(sql, function (err, result) {
if (err) throw err;
console.log("1 record inserted");
});
});
```

Output:

```
D:\Node-js>node Database-connect.js
Connected!
1 record inserted
```

Updating a record in the table

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "root",
database: "test"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
var sql = "UPDATE employees2 SET city = 'Pune' WHERE city = 'US'";
con.query(sql, function (err, result) {
if (err) throw err;
console.log(result.affectedRows + " record(s) updated");
});
});
```

Output:

```
D:\Node-js>node Database-connect.js
Connected!
1 record(s) updated
```

Fetching record from the table

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "root",
database: "test"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
con.query("SELECT * FROM employees2", function (err, result) {
if (err) throw err;
console.log(result);
});
});
```

Output:

```
D:\Node-js>node Database-connect.js
Connected!
[ RowDataPacket { id: 1, name: 'Allen Turin', age: 27, city: 'Pune' } ]
```

Deleting record from the table

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "root",
database: "test"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
var sql = "DELETE FROM employees2 WHERE city = 'pune'";
con.query(sql, function (err, result) {
if (err) throw err;
console.log("Number of records deleted: " + result.affectedRows);
});
});
```

Output:

```
D:\Node-js>node Database-connect.js
Connected!
Number of records deleted: 1
```

Thank You

`