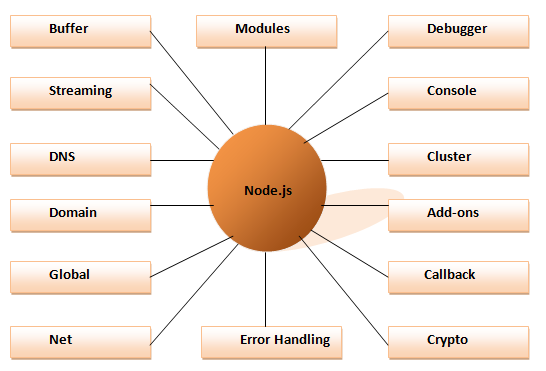
**Node JS**

* Node.js is a **cross-platform** runtime environment and library for running JavaScript applications outside the browser.
* It is used for creating **server-side** and networking web applications.
* It is open source and free to use. It can be downloaded from this link [**https://nodejs.org/en/**](https://nodejs.org/en/) **.**
* Many of the basic modules of Node.js are written in JavaScript.
* Node.js is mostly used to run real-time server applications.

* Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications.
* Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
* Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.

**Different parts of Node.js**



**Run Node Js Code**

**>> node** <yourFile**.js**> **File extension** **<anyName>.js**

**Node.js Console**

* The Node.js console module provides a simple debugging console similar to JavaScript console mechanism provided by web browsers.
* There are three console methods that are used to write any node.js stream:

1. console.log()
2. console.error()
3. console.warn()

**Node.js Global Objects**

* Node.js global objects are global in nature and available in all modules.
* You don't need to include these objects in your application; rather they can be used directly.
* These objects are modules, functions, strings and object etc.
* Some of these objects aren't actually in the global scope but in the module scope.

**Global Objects:**

* \_\_dirname
* \_\_filename
* Console
* Process
* Buffer
* setImmediate()
* setInterval()
* setTimeout()
* clearImmediate()
* clearInterval()
* clearTimeout()

**Node.js Timer**

Node.js Timer functions are **global functions**. You don't need to use require() function in order to use timer functions. Let's see the list of timer functions.

**Set timer functions:**

* **setInterval():** It is used to define a time interval.
* **setTimeout():** It is used to execute a one-time callback.

**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

**Node.js File System (FS)**

In Node.js, file I/O is provided by simple wrappers around standard POSIX functions. Node File System **(fs)** module can be imported using following syntax:

**Note : const** fs = **require(“**fs**”);**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **readFileSync(<path>,<encoding>)** | Synchronously reads the entire contents of a file.  **Return : String file data** |
| **readFile(<path>,<encoding>,<callBackFun>)** | Asynchronously reads the entire contents of a file. |
| **writeFileSync(<path>,<data>)** | Synchronously write data into file. |
| **writeFile(<path>,<data>,<callBackFun>)** | ASynchronously write data into file |
| **appendFileSync(<path>,<data>)** | Synchronously append data into file. |
| **appendFile(<path>,<data>,<callBackFun>)** | ASynchronously append data into file. |
| **copyFileSync(<srcFilePath>,<newFilePath>)** | Copy one file data to another file Synchronously. |
| **copyFile(<srcFilePath>,<newFilePath>)** | Copy one file data to another file Asynchronously. |
| **unLinkSync(<path>)** | Delete the file Synchronously. |
| **unLink(<path>,<callBackFunction>)** | Delete the file ASynchronously |

**Examples :**

**readFileSync(<path>,<encoding>)**

//Synchronously reads the entire contents of a file.

**const** filePath = "<filePath>";

**const** userData **= fs.readFileSync(**filePath**,"**utf-8**");**

console.log**("**File Read Complete ........**");**

console.log**("**Process End ........**"**);

**readFile(<path>,<encoding>,<callBackFun>)**

//Asynchronously reads the entire contents of a file.

userData = **fs.readFile**(filePath,"utf-8",(error,data) => {

**if**(error) **throw** error;

console.log("File Read Complete ....");

});

console.log**("**Process End ........**"**);

**writeFileSync(<path>,<data>)**

//Synchronously write data into file.

**fs.writeFileSync**("Node\_Class\_Code\\File\\Sample.txt","Test Records");

console.log("Process End .....");

**writeFile(<path>,<data>,<callBackFun>)**

//ASynchronously write data into file.

**fs.writeFile**(filePath,"Sample data",(error) => {

**if**(error) **throw** error;

console.log("File Write Complete .....");

});

console.log("Process End .....");

**appendFileSync(<path>,<data>)**

//Synchronously Append data into file

**fs.appendFileSync**(filePath,"Append Data");

console.log("Process End .....");

**appendFile(<path>,<data>,<callBackFun>)**

//ASynchronously Append data into file

**fs.appendFile**(filePath,"Async Append data",(error) => {

**if**(error) **throw** error;

console.log("Append Process Complete .....");

});

console.log("Process End .....");

**copyFileSync(<srcFilePath>,<newFilePath>)**

//Copy one file data to another file

**fs.copyFileSync**(sourceFilePath,"newFile.txt");

console.log("Process End .....");

**copyFile(<srcFilePath>,<newFilePath>)**

//Copy one file data to another file ASynchronously

**fs.copyFile**(sourceFilePath,"newFile.txt",(error) => {

**if**(error) **throw** error;

console.log("Success ........");

});

console.log("Process End .....");

**unLinkSync(<path>)**

//Delete file Synchronously

**fs.unlinkSync**("Node\_Class\_Code\\File\\newFile.txt");

console.log("Process End .......");

**unLink(<path>,<callBackFunction>)**

//Delete File ASynchronously

fs.unlink("Node\_Class\_Code\\File\\newFile.txt",(error) => {

if(error) throw error;

console.log("Deleted .........");

});

console.log("Process End .......");

**Node.js Streams**

Streams are the objects that facilitate you to read data from a source and write data to a destination. There are four types of streams in Node.js:

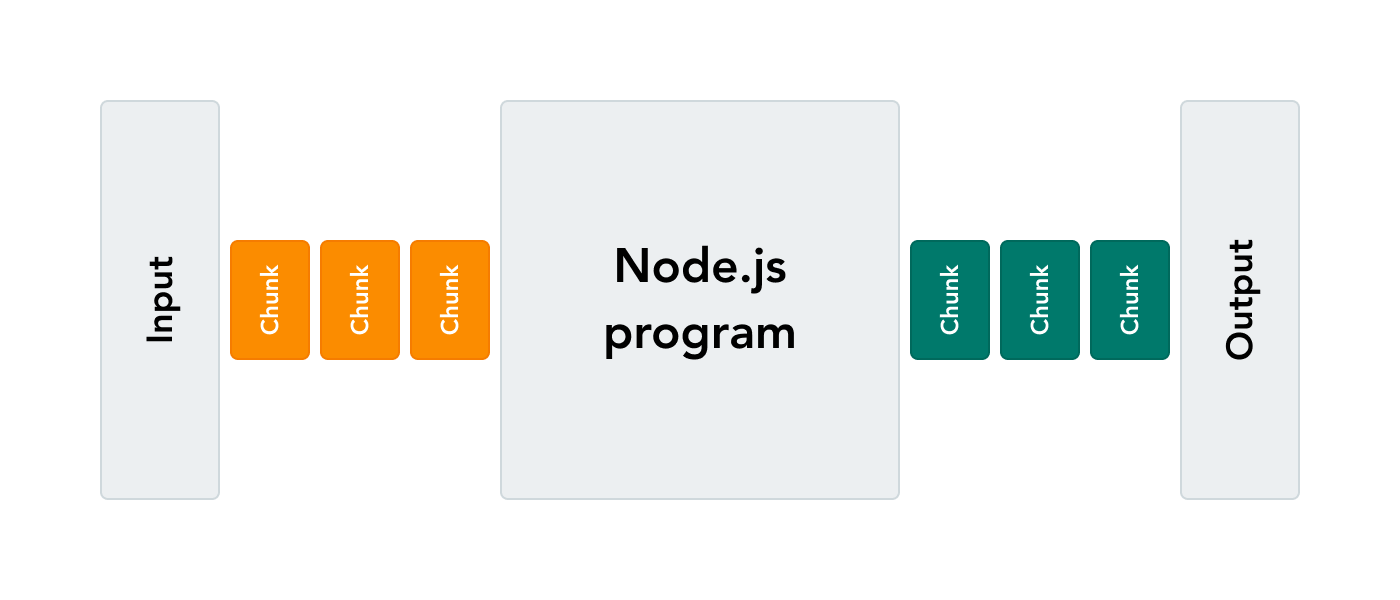
* **Readable:** This stream is used for read operations.

* **Writable:** This stream is used for write operations.
* **Duplex:** This stream can be used for both read and write operations.
* **Transform:** It is type of duplex stream where the output is computed

according to input.

Each type of stream is an Event emitter instance and throws several events at different times. Following are some commonly used events:

* **Data:** This event is fired when there is data available to read.
* **End:** This event is fired when there is no more data available to read.
* **Error:** This event is fired when there is any error receiving or writing data.
* **Finish:** This event is fired when all data has been flushed to un



**Example 1:**

**Note : const** fs = **require(“**fs**”);**

**let** filePath = “your File path”;

**const** readStream = **fs.createReadStream(**filePath,”utf-8”**);**

//handle data event

readStream**.on(**“data”,(data) => {

console.log(data);

}**);**

//handle error event

readStream**.on(**“error”,(error) => {

**if**(error) **throw** error;

}**);**

**Example 2:**

**let** filePath = “your File path”;

**const** writeStream = **fs.createWriteStream(**filePath**);**

//handle finish event

readStream**.on(**“finish”,() => {

console.log(“file write finish”);

}**);**

//handle error event

writeStream**.on(**“error”,(error) => {

**if**(error) **throw** error;

}**);**

**Node.js Events**

In Node.js applications, Events and Callbacks concepts are used to provide concurrency. **As Node.js applications are single threaded and every API of Node js are asynchronous**. So it uses async function to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and after the completion of any task, it fires the corresponding event which signals the event listener function to get executed.

**Event Driven Programming**

Node.js uses event driven programming. It means as soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for event to occur. It is the one of the reason why Node.js is pretty fast compared to other similar technologies.

There is a main loop in the event driven application that listens for events, and then triggers a callback function when one of those events is detected.

## Difference between Events and Callbacks:

Although, Events and Callbacks look similar but the differences lies in the fact that callback functions are called when an asynchronous function returns its result where as event handling works on the observer pattern. Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through events module and EventEmitter class which is used to bind events and event listeners.

**Note :** **const** events = **require(**“events”**);**

**Methods**

1) **<events>.on(<eventName>,<callBackFunction>)**

2) **<events>.once(<eventName>,<callBackFunction>)**

3) **<events>.emit(<eventName>)**

**(ex)** //create class

**class** MyEvent **extends events.EventEmitter** {}

//create object for MyEvent class

**const** myEvent = **new** MyEvent();

myEvent**.on(**“myEvent”,() => {

console.log(“ My Event is work ”);

}**);**

myEvent**.on(“**myEvent**”),**() => {

console.log(“ My Second Event is work ”);

});

//invoke the event

myEvent**.emit(**“myEvent”**);**

**(ex)** //once method

myEvent**.once(**“myEventOnce”,() => {

console.log(“ My Event Once work ”);

}**);**

//invoke the event

myEvent**.emit(“**myEventOnce**”);**

myEvent.**emit(“**myEventOnce**”);**

**Node.js Web Module**

**What is Web Server**

Web Server is a software program that handles **HTTTP** requests sent by HTTP clients like web browsers, and returns web pages in response to the clients. Web servers usually respond with html documents along with **images, style sheets and scripts**.

Most of the web server support server side scripts using scripting language or redirect to application server which perform the specific task of getting data from database, perform complex logic etc. and then sends a result to the **HTTP client through the Web server**.

**Create Simple Web Server**

**Method : createServer(<callBackFunction>);**

**listen(<portNumber>,<callBackFunction>?);**

**Note : const** http = **require(**“http”**);**

**(ex)** //require the http module from node core modules

**const** http = **require(**“http”**);**

//create web server

**const** server = http**.createServer(**(request,response) => {

//send the response to client

response**.end(**“ Welcome to Node Js Server ”**);**

}**);**

//listen the server

server**.listen(**3000,() => {

console.log(“ Server is Running . . . ”);

}**);**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000**

**Create Simple Web Server and send HTML RESPONSE**

**Method : writeHead(<statusCode>,<object>);**

**(ex)** //require the http module from node core modules

**const** http = **require(**“http”**);**

//create web server

**const** server = http**.createServer(**(request,response) => {

//set response header

response**.writeHead(**200,{

“Content-type” : “text/html”,

“MyHeader” : “Sample Text”

}**);**

//send the HTML response to client

response**.end(**“ Welcome to Node Js Server ”**);**

}**);**

//listen the server

server**.listen(**3000,() => {

console.log(“ Server is Running . . . ”);

}**);**

**Run : node <yourFileName>.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000**

**Send HTML Page to client (Use Same Directory)**

**Note : 1st create HTML File called index.html**

**FILE : index.html**

**<html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1>** This is My First Web Page **</h1>**

**</body>**

**</html>**

**FILE : app.js**

**const** http = **require(**"http"**);**

**const** fs = **require(**"fs"**);**

**let** template = fs**.readFileSync(**"index.html"**);**

**const** server = http**.createServer**((request,response) => {

//set response header and status code

response**.writeHead(**200,{

"Content-Type" : "text/html",

"MyHeader" : "Test Text"

}**);**

//sending html response to client

response**.write(**template**);**

response**.end();**

}**); Run : node app.js**

server**.listen(**3000,() => {console.log("Server is running ....");}**);**

**Create Simple Routing**

**const** http = **require("**http**");**

**const** server **=** http.**createServer(**(request,response) => {

//print the new Request url

console.log("Request URL : ",request**.url**);

//get request url

const url = request**.url.toLowerCase();**

//create simple routing

**if**(url == "/" || url == "/home") response.end("Home Page"**);**

**else if**(url == "/about") response**.end("**About Page"**);**

**else if**(url == "/contact") response**.end**("Contact Page"**);**

**else** response**.end(**"Page not found : ERROR CODE 404"**);**

}**);**

**server.listen(**3000,() => {console.log("Server is running ....");}**);**

**Run : node app.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000/**

**NEXT 🡺 localhost:3000/home output :** Home Page

**NEXT 🡺 localhost:3000/about output :** About Page

**NEXT 🡺 localhost:3000/contact output :** Contact Page

**NEXT 🡺 localhost:3000/randomText output :** Page not Fount : 404

**Create Simple Routing and send HTML Response**

**(Use Same Directory)**

**FILE : index.html**

**<html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1>** Home Page **</h1>**

**</body>**

**</html>**

**FILE : about.html**

**<html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1>** About Page **</h1>**

**</body>**

**</html>**

**FILE : contact.html**

**<html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1>** Contact Page **</h1>**

**</body>**

**</html>**

**FILE : app.js**

**const** http = **require(**"http"**);**

**const** fs = **require(**"fs"**);**

**let** homePageHTML = fs**.readFileSync(**"index.html"**);**

**let** aboutPageHTML = fs**.readFileSync(**"about.html"**);**

**let** contactPageHTML = fs**.readFileSync(**"contact.html"**);**

**const** server = http**.createServer**((request,response) => {

**const** url = request**.url.toLowerCase();**

**if(**url == “/” || url == “/home”**){**

//set response header and status code

response**.writeHead(**200,{

"Content-Type" : "text/html",

}**);**

response**.end(**homePageHTML**);**

**}**

**else if(**url == “/about”**){**

//set response header and status code

response**.writeHead(**200,{

"Content-Type" : "text/html",

}**);**

response**.end(**aboutPageHTML**);**

**}**

**else if(**url == “/contact”**){**

//set response header and status code

response**.writeHead(**200,{

"Content-Type" : "text/html",

}**);**

response**.end(**contactPageHTML**);**

**}**

**else {**

//set response header and status code

response**.writeHead(**404,{

"Content-Type" : "text/html",

}**);**

response**.end(**“<p> Page not Found : ERROR CODE 404”**);**

**}**

}**);**

server**.listen(**3000,() => {console.log("Server is running ....");}**);**

**Run : node app.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000/**

**NEXT 🡺 localhost:3000/home output :** Home Page

**NEXT 🡺 localhost:3000/about output :** About Page

**NEXT 🡺 localhost:3000/contact output :** Contact Page

**NEXT 🡺 localhost:3000/randomText output :** Page not Fount : 404

**Node.js URL Module**

# The URL module splits up a web address into readable parts.

# A URL string is a structured string containing multiple meaningful components. When parsed, a URL object is returned containing properties for each of these components.

# Note : const url = require(“url”);

# Methods :

# 1) url.parse(<url>) 🡺 Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

# 2) url.format(<urlObject>) 🡺 The url.format() method returns a formatted URL string derived fromurlObject.

# Method url.parse():

# (ex) const url = require(“url”);

# let o = url.parse('http://localhost:8080/default.htm?year=2017&month=february');

# console.log(o.protocol);

# console.log(o.hostname);

# console.log(o.pathname);

# console.log(o.query);

# Method url.format():

# (ex) const url = require(“url”);

# let newURL = url.format({

# protocol : “https”,

# hostname : “localhost”,

# pathname : “sample/path”,

# query : {

# userName : “Surya”,

# password : “@1234.com”

# }

# });

# console.log(newURL);

# Using URL Class Methods

# The URL interface represents an object providing static methods used for creating object URLs.

# Property:

# href 🡺 return full url.

# pathname 🡺 return path name of url.

# hostname 🡺 return host name of url.

# port 🡺 return port number of url.

# search 🡺 return query string or search string of url

# protocol 🡺 return protocol of url (ex) http / https

# searchParms 🡺 return parms of url (ex) age = 10

# Methods

# searchParms.get(<string>) 🡺 Returns the first value associated to the given search parameter.

# searchParms.set(<key>,<value>) 🡺 Sets the value associated to a given search parameter to the given value. If there were several values, delete the others.

# searchParms.append(<key>,<value>) 🡺 Appends a specified key/value pair as a new search parameter.

# searchParms.has(<string>) 🡺 check specific key in url.

# searchParms.delete(<string>) 🡺 delete specific key in url.

# searchParms.keys() 🡺 return all parm keys

# searchParms.value() 🡺 return all parm values

# searchParms.entries() 🡺 return all keys and values.

# searchParms.getAll(<string>) 🡺 return all value of given key.

# (ex)

# const myURL = new URL('http://localhost:8080/default.htm?year=2017&month=february');

# console.log(myURL.port);

# console.log(myURL.hostname);

# console.log(myURL.protocol);

# console.log(myURL.searchParms.get(“year”));

**Node.js Query String**

The Node.js Query String provides methods to deal with query string. It can be used to **convert query string into JSON object** and vice-versa.

To use query string module, you need to use **require('**querystring**')**.

The two **important methods** are given below.

**Note : const** querystring = **require(**“querystring”**);**

**Methods :**

**parse(<queryString>,<separator>?) 🡺** The querystring.parse() method parses a URL query string (str) into a collection of key and value pairs**.**

**stringify(<object>,<separator>?) 🡺** method produces a URL query string from a given obj by iterating through the object's "own properties".

**Method : parse()**

**(ex) const** querystring = **require(**“querystring”**);**

**let** user = querystring**.parse(**“userName=Surya&userPassword=1234”**);**

console.log(user.userName);

console.log(user.userPassword);

**Method : stringify()**

**ex) const** querystring = **require(**“querystring”**);**

**let** user = querystring**.parse(**“userName=Surya&userPassword=1234”**);**

**let** stringURL = querystring**.stringify(**user**);**

console.log(typeof user);

console.log(typeof stringURL);

console.log(stringURL);

**Node.js Path**

The Node.js path module is used to handle and transform files paths. module can be imported by using the following syntax:

**Note : const** path = **require(**“path”**);**

**Methods**

**join([...path]) 🡺** The path.join() method joins all given path segments together using the **platform-specific** separator as a delimiter, then normalizes the resulting path.

**basename(<filePath>) 🡺** return basename of path.

**dirname(<filePath>) 🡺** return direction name of path.

**extname(<filePath>) 🡺** return file extension name of path.

**isAbsolute(<filePath>) 🡺** return boolean value if path is absolute.

**parse(<filePath>) 🡺** return path object.

**format(<object>) 🡺** return string path.

**(ex)**

**const** path **= require("**path**");**

**let** filePath **=** "C:\\Web\_Tech\\Node\_JS\\Node\_Class\_Code\\coreModules\\pathModule.js";

console.log(path**.isAbsolute**(filePath));

console.log(path**.join**(**\_\_dirname**,"newFile.txt"));

console.log(path**.extname**(filePath));

console.log(path**.dirname**(filePath));

console.log(path**.basename**(filePath));

console.log(path**.parse**(filePath));

**GET Form Data (Use Same Directory)**

**FILE : form.html**

**<html>**

**<head> <title>** Form **</title></head>**

**<body>**

**<form action="**/user**" method="**POST**">**

**<label for = "**userName**" >** Enter your Name : **</label>**

**<input type = "**text**" name = "**userName**" id = "**userName**">**

**<br><br>**

**<label for = "**userAge**" >** Enter your Age : **</label>**

**<input type = "**number**" name = "**userAge**" id = "**userAge**">**

**<br><br>**

**<label for = "**userEmail**" >** Enter your Email : **</label>**

**<input type = "**text**" name = "**userEmail**" id = "**userEmail**">**

**<br><br>**

**<label for = "**userPassword**" >** Enter your Password : **</label>**

**<input type = "**password**" name = "**userPassword**" id = "**userPassword**">**

**<br><br>**

**<label for = "**userDOB**" >** Enter your DOB : **</label>**

**<input type = "**date**" name = "**userDOB**" id = "**userDOB**">**

**<br><br>**

**<label >** Gender : **</label>**

**<input type="radio" name = "gender" id = "male" value = "male">**

**<label for = "male">** Male **</label>**

**<input type= "radio" name = "gender" id = "male" value = "female">**

**<label for = "**male**">** Female **</label>**

**<br><br>**

**<label>** Courses : **</label>**

**<input type="**checkbox**" value="**C**" id = "**C**" name = "**course**">**

**<label for = "**C**">** C **</label>**

**<input type="**checkbox**" value="**CPP**" id = "**CPP**" name = "**course**">**

**<label for = "CPP"> C++ </label>**

**<input type="**checkbox**" value="**Java**" id = "**Java**" name = "**course**">**

**<label for = "**Java**">** Java **</label>**

**<br><br>**

**<label>** Hobbies : **</label>**

**<select name = "**hobbies**">**

**<option value = "" selected >** Select **</option>**

**<option value = "**footBall**" >** Foot Ball **</option>**

**<option value = "**cricket**" >** Cricket **</option>**

**<option value = "**volley Ball**" >** Volley Ball **</option>**

**</select>**

**<br><br>**

**<label for = "**address**">** Address : **</label> <br>**

**<textarea name="address" id="address" cols="30" rows="2"></textarea**

**<br><br>**

**<input type = "**submit**" value = "**Submit**">**

**<input type = "**reset**" value = "**Reset**">**

**</form>**

**</body>**

**</html>**

**FILE : userDetails.html**

**<html>**

**<head> <title> Document </title> </head>**

**<body>**

**<p>** Name : {{%userName%}} **</p>**

**<p>** Age : {{%userAge%}} **</p>**

**<p>** Email : {{%userEmail%}} **</p>**

**<p>** Password : {{%userPassword%}} **</p>**

**<p>** DOB : {{%userDOB%}} **</p>**

**<p>** Gender : {{%userGender%}} **</p>**

**<p>** Course : {{%course%}} **</p>**

**<p>** Hobbies : {{%userHobbies%}} **</p>**

**<p>** Address : {{%userAddress%}} **</p>**

**</body>**

**</html>**

**FILE : app.js**

**const** http **= require("**http**");**

**const** fs **= require("**fs**");**

**const** querystring **= require("**querystring**");**

**const** path **= require("path");**

**let** formHTML **= fs.readFileSync(path.join(\_\_dirname,"form.html"));**

**let** userDetailsHTML **= fs.readFileSync(path.join(\_\_dirname,"userDetails.html"));**

**const** server **=** http**.createServer();**

**server.on("**request**",(**request,respose**) => {**

**let** url **=** request**.url;**

**if(**url == "/"**){**

respose**.writeHead(**200,{"Content-Type" : "text/html"}**);**

respose**.end(**formHTML**);**

**}**

**if(**url == "/user"**){**

**request.on(**"data",(chunk) => {

**let** data **= Buffer.from(**chunk**).toString();**

**let** user **=** querystring**.parse(**data**,"**&**");** //return object

//change placeholder to value

userDetailsHTML **= String(**userDetailsHTML**)**

**.replace(**"{{%userName%}}",user.userName**)**

**.replace(**"{{%userAge%}}",user.userAge**)**

**.replace(**"{{%userEmail%}}",user.userEmail**)**

**.replace(**"{{%userPassword%}}",user.userPassword**)**

**.replace(**"{{%userDOB%}}",user.userDOB**)**

**.replace(**"{{%userGender%}}",user.gender**)**

**.replace(**"{{%course%}}",user.course**)**

**.replace(**"{{%userHobbies%}}",user.hobbies**)**

**.replace(**"{{%userAddress%}}",user.address**);**

**respose.end(userDetailsHTML);**

**});** //result.on end

**}** // if end

**});** //server.on end

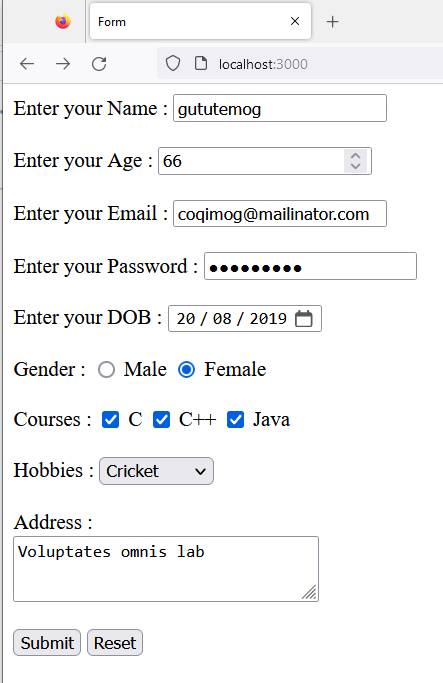
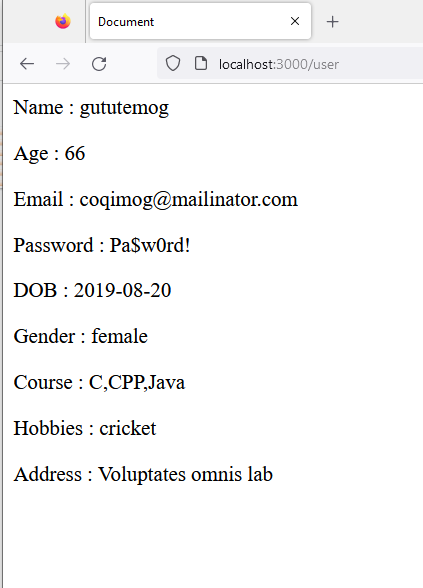
**server.on(**"error",(error) => {console.log(error);}**);**

**server.listen(**3000,() => {console.log("Server is Running ...."); }**);**

**Run : node app.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000/**

**NEXT 🡺 Click Submit Button**

**output :**

# Node.js - NPM

**Node Package Manager (NPM) provides two main functionalities:**

* Online repositories for node.js packages/modules which are searchable on [search.nodejs.org](https://search.nodejs.org)
* Command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

# Installing Modules using NPM

**Syntax : >> npm install <packageName>**

**step 1 : open cmd and type given command**

**(ex) npm install express**

# Global vs Local Installation

By default, NPM installs any dependency in the local mode. Here local mode refers to the package installation in node\_modules directory lying in the folder where Node application is present. Locally deployed packages are accessible via require() method. For example, when we installed express module, it created node\_modules directory in the current directory where it installed the express module.

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

**(ex) npm install express -g**

## Uninstalling a Module

**Syntax : >> npm uninstall <packageName>**

**step 1 : open cmd and type given command**

**(ex) npm uninstall express**

## Updating a Module

Update package.json and change the version of the dependency to be updated and run the following command.

**Syntax : >> npm update <packageName>**

**step 1 : open cmd and type given command**

**(ex) npm update express**

## Create a Module

Creating a module requires package.json to be generated. Let's generate package.json using NPM, which will generate the basic skeleton of the package.json.

**Syntax : >> npm init**

**step 1 : open cmd and type given command**

**(ex) npm init**

**Node.Js Create Connection with MySQL**

You have to install **MySQL driver to access a MySQL database** with Node.js. Download MySQl module from **npm**.

**step 1 : open cmd and type given command**

**(ex) npm install mysql2**

## Create Connection :

//create connection with mysql

**const** mysql= **require("**mysql2**");**

**const** connection **=** mysql**.createConnection(**{

**host : "**localhost**",**

**user : "**<mysql\_userName>**",**

**password : "**<mysql\_password>**"**

}**);**

//use connect method

connection**.connect(**(error) => {

**if(**error**) throw** error;

console.log**("**DataBase Connected ....**");**

}**);**

## Create Database :

//create database using node js

**const** mysql = **require(**"mysql2"**);**

**const** connection = mysql**.createConnection(**{

**host :** "localhost",

**user :** "root",

**password :** "root",

}**);**

**connection.connect(**(error) => {

**if(**error**) throw** error;

//CREATE DATABASE QUERY STRING

**let** queryString = "CREATE DATABASE NODE\_MYSQL";

connection**.query**(queryString,(error,result) => {

**if(**error**)** **throw** error;

console.log("DataBase Created Successfully ......");

}**);**

//USE DATABASE QUERY STRING

queryString = "USE NODE\_MYSQL";

connection**.query(**queryString,(error,result) => {

**if(**error**) throw** error;

console.log("Database Changed .....");

}**);**

}**);**

## Create Table :

//create table using node js

**const** mysql = **require(**"mysql2"**)**;

**const** connection = mysql.createConnection({

**host :** "localhost",

**user :** "root",

**password :** "root",

**database :** "node\_mysql"

});

connection**.connect(**(error) => {

if(error) throw error;

**let** queryString = "CREATE TABLE USERS (ID INT, NAME VARCHAR(30),

AGE INT , SALARY INT, EMAIL VARCHAR(255) )";

connection**.query(**queryString,(error,result) => {

if(error) throw error;

console.log("Table Created Successfully ......");

}**);**

}**);**

## Insert Single Record into Database :

//insert single record into mysql

**const** mysql = **require(**"mysql2"**);**

**const** connection = mysql**.createConnection(**{

**host :** "localhost",

**user :** "root",

**password :** "root",

**database :** "node\_mysql"

}**);**

connection**.connect(**(error) => {

**if(**error**) throw** error;

**let** queryString = "INSERT INTO USERS ( ID, NAME, AGE, SALARY, EMAIL ) VALUES ( 1, 'SURYA', 20, 1000, 'surya@gmail.com' )";

connection**.query(**queryString,(error,result) => {

**if(**error**) throw** error;

console.log("Record Insert Successfully .....");

}**);**

}**);**

## Insert Multiple Records into Database :

//insert multiple records into database

**const** mysql = **require(**"mysql2"**);**

**const** connection = mysql**.createConnection(**{

**host :** "localhost",

**user :** "root",

**password :** "root",

**database :** "node\_mysql"

}**);**

connection**.connect(**(error) => {

**if(**error**)** **throw** error;

//create temp user records

**let** userRecords = [

[2, "Ram", 30, 3000, "ram@gmail.com"],

[3, "Sam", 50, 4000, "sam@gmail.com"],

[4, "Kumar", 20, 5000, "kumar@gmail.com"],

[5, "Arul", 11, 8000, "arul@gmail.com"],

[6, "Sai", 18, 10000, "sai@gmail.com"]

];

**let** queryString = "INSERT INTO USERS ( ID, NAME, AGE, SALARY, EMAIL ) VALUES ? ";

connection**.query(**queryString,[userRecords],(error,result) => {

**if(**error**)** **throw** error;

console.log("Records Insert Successfully .....");

}**);**

}**);**

## Fetch All Records From Database :

//select all records from database

**const** mysql = **require(**"mysql2"**);**

**const** connection = mysql**.createConnection(**{

**host :** "localhost",

**user :** "root",

**password :** "root",

**database :** "node\_mysql"

}**);**

connection**.connect(**(error) => {

**if(**error**)** **throw** error;

**let** queryString = "SELECT \* FROM USERS";

connection**.query(**queryString,(error,result) => {

**if(**error**)** **throw** error;

console.log(result);

}**);**

}**);**

**output :**

[

{ ID: 1,NAME: 'SURYA',AGE: 20,SALARY: 1000,EMAIL:'surya@gmail.com' },

{ ID: 2, NAME: 'Ram', AGE: 30, SALARY: 3000, EMAIL: 'ram@gmail.com' },

{ ID: 3, NAME: 'Sam', AGE: 50, SALARY: 4000, EMAIL: 'sam@gmail.com' },

{ ID: 4, NAME: 'Kumar', AGE: 20, SALARY: 5000, EMAIL: 'kumar@gmail.com'},

{ ID: 5, NAME: 'Arul', AGE: 11, SALARY: 8000, EMAIL: 'arul@gmail.com' },

]

## Delete Record From Database :

//delete records from database

connection**.connect(**(error) => {

**if(**error**)** throw error;

**let** queryString = "DELETE FROM USERS WHERE ID = 1";

connection**.query(**queryString,(error,result) => {

**if(**error**)** **throw** error;

console.log("Record Deleted Successfully .....");

}**);**

}**);**

## Delete Record From Database :

connection**.connect(**(error) => {

**if(**error**)** throw error;

**let** queryString = "UPDATE USERS SET NAME = 'TEMP' WHERE ID = '2'";

connection**.query(**queryString,(error,result) => {

**if(**error**)** **throw** error;

console.log("Record Update Successfully .....");

}**);**

}**);**

## Update Record :

//update record

**const** mysql = **require(**"mysql2"**);**

**const** connection = mysql**.createConnection(**{

**host :** "localhost",

**user :** "root",

**password :** "root",

**database :** "node\_mysql"

}**);**

connection**.connect(**(error) => {

**if(**error**)** **throw** error;

**let** queryString = "UPDATE USERS SET NAME = 'TEMP' WHERE ID = '2'";

connection**.query(**queryString,(error,result) => {

**if(**error**) throw** error;

console.log("Record Update Successfully .....");

}**);**

}**);**