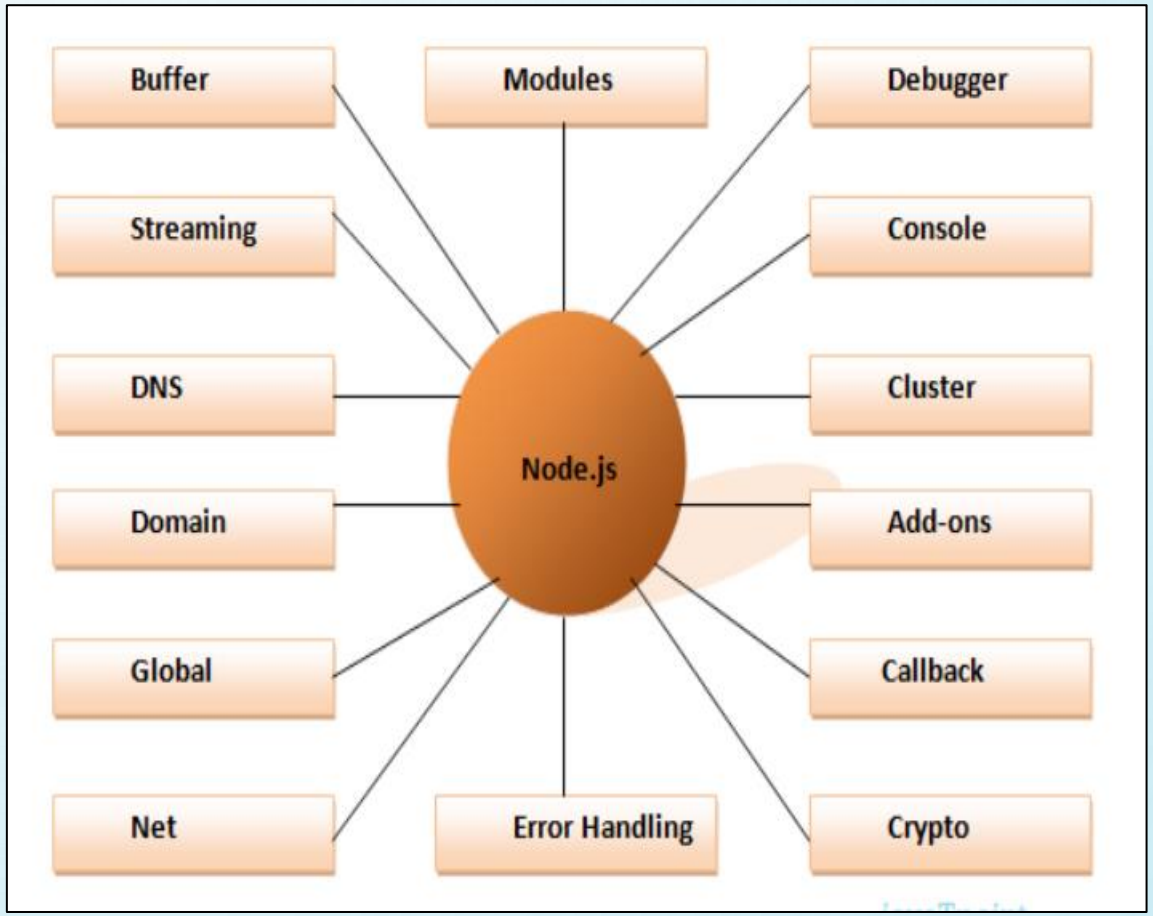
FULL STACK WEB DEVELOPMENT

Module: 4 Web Server – Node JS

Getting started with Node.js – installing Node.js – working with Node Packages, creating Node.js application – using Events, Listeners, Timers and Callbacks in Node.js – Implementing Event emitter, implementing Callbacks – Accessing the File System from Node.js – implementing HTTP services in Node.js – saving time with Express – the request and response objects – Form Handling – Sending Client Data to Server, Form Handling with Express – Cookies and Sessions.

Node.JS – Introduction

- 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.
- 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 – Features

- **Extremely fast**: Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
- **I/O is Asynchronous** and **Event Driven**: All APIs of Node.js library are asynchronous i.e. non-blocking. So 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. It is also a reason that it is very fast.
- **Single threaded**: Node.js follows a single threaded model with event looping.
- **Highly Scalable**: Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
- **No buffering**: Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
- **Open source**: Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
- **License**: Node.js is released under the MIT license.

Node.js – Environment

- To install and setup an environment for Node.js, you need the following two software available on your computer:

  - **Text Editor**.
  - **Node.js Binary installable**

Installation help URL: https://www.javatpoint.com/install-nodejs

Node.js applications are categorized as console-based applications and web-based applications.

https://nodejs.org

Node.js – Console

- Type and save the code with the extension of **.js**
- Use the command '**node**' to execute the program.

E.g.,

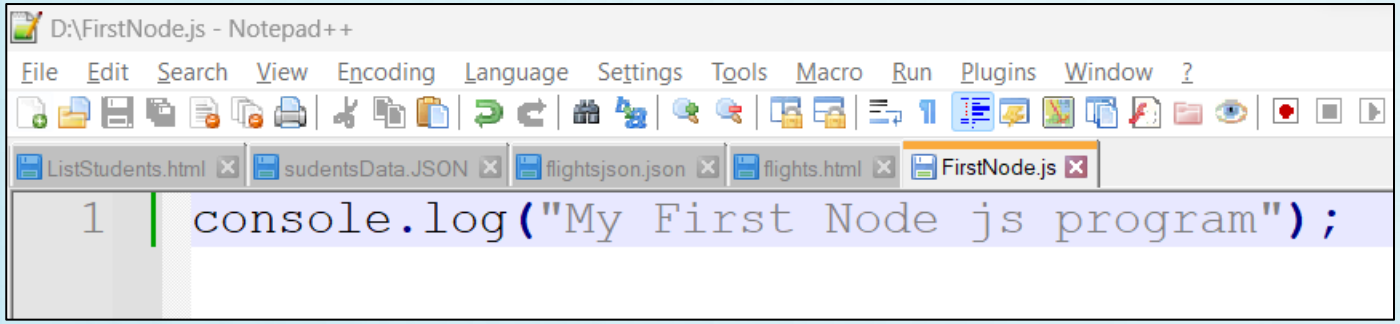
Type and save the following as **FirstNode.js**

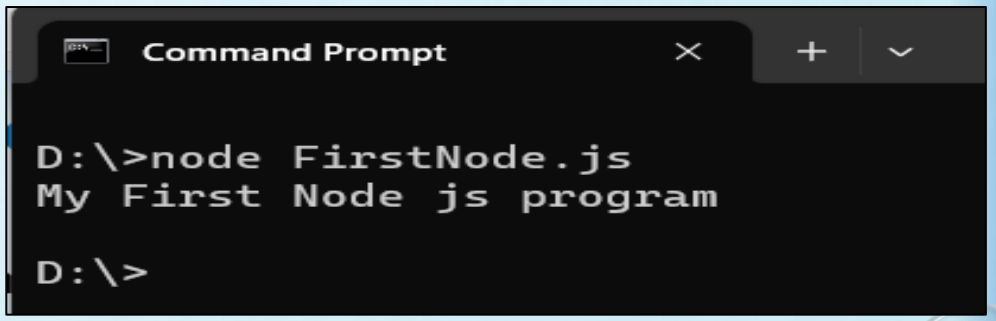
console.log("My First Node js program");

Execution 'open the command prompt and give the following command':

>**node FirstNode.js**

My First Node js program





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:
  - ✓ **console.log()** - is used to display simple message on console.
  - ✓ **console.error()** - is used to render error message on console.
  - ✓ **console.warn()** - is used to display warning message on console.

```
console.error(new Error('Hell! This is a wrong method.'));
```

Node.js command prompt

```
C:\Users\javatpoint1\Desktop>node console_example3.js
[Error: Hell! This is a wrong method.]

C:\Users\javatpoint1\Desktop>_
```

```
const name = 'John';

console.warn(`Don't mess with me ${name}! Don't mess with me!`);
```

Node.js command prompt

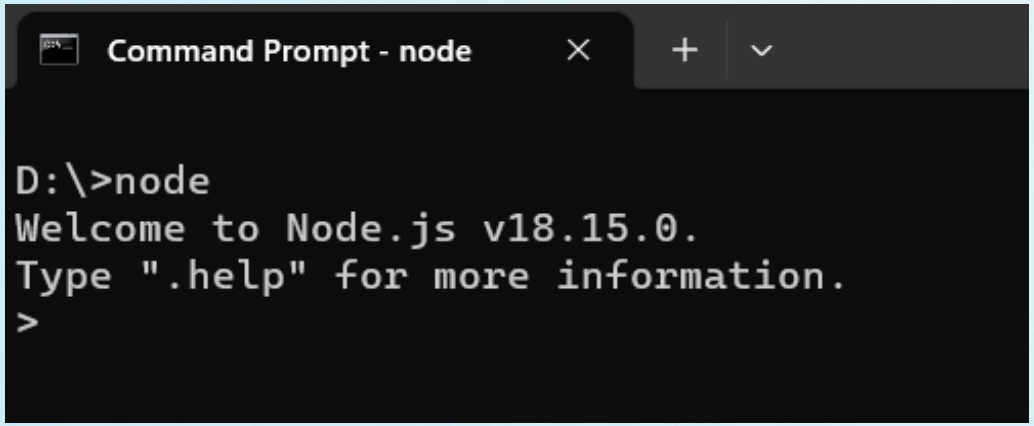
```
C:\Users\javatpoint1\Desktop>node console_example4.js
Don't mess with me John! Don't mess with me!

C:\Users\javatpoint1\Desktop>_
```

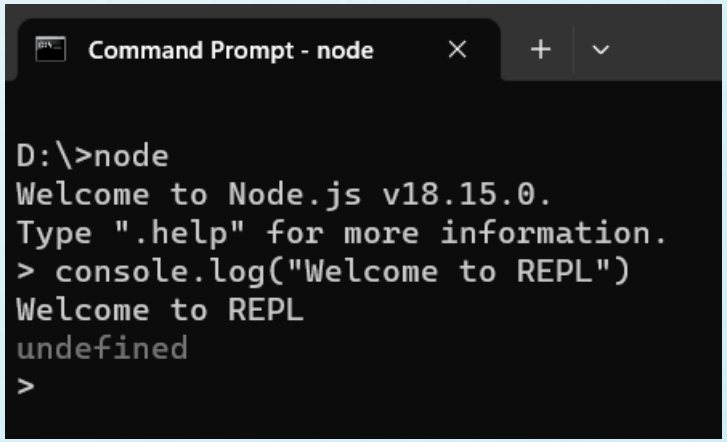
Node.js – REPL

The term **REPL** stands for **Read Eval Print** and **Loop**. It specifies a computer environment like a window console or a Unix/Linux shell where you can enter the commands and the system responds with an output in an interactive mode.

- **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 in command prompt to start REPL





Node.js – REPL

Command Prompt

```
D:\>node
Welcome to Node.js v18.15.0.
Type ".help" for more information.
> 5*9
45
> 8+6/3
10
> 8+6/3.0
10
>
(To exit, press Ctrl+C again or Ctrl+D or type .exit)
>

D:\>
```

Command Prompt - node

```
45
> 8+6/3
10
> 8+6/3.0
10
>
(To exit, press Ctrl+C again or Ctrl+D or type .exit)
>

D:\>node
Welcome to Node.js v18.15.0.
Type ".help" for more information.
> var x=0
undefined
> do {
... x++;
... cosole.log("x : "+x);
... }while(x<5);
Uncaught ReferenceError: cosole is not defined
> do {
... x++;
... console.log("x : "+x);
... }while(x<=5);
x : 2
x : 3
x : 4
x : 5
x : 6
undefined
>
```

Node.js – Web-based Example

- **Import required modules**: The "require" directive is used to load a Node.js module.
- **Create server**: You have to establish a server which will listen to client's request similar to Apache HTTP Server.
- **Read request and return response**: Server created in the second step will read HTTP request made by client which can be a browser or console and return the response.
- Sends an HTTP response status code of **200** (OK), Sets the response header to indicate that the content type is **HTML**, so the browser knows to interpret the response as an HTML document.

```
var http = require("http");
http.createServer(function (request, response) {
 // Send the HTTP header
  // HTTP Status: 200 : OK
  // Content Type: text/plain
  response.writeHead(200, {'Content-Type': 'text/plain'});
  // Send the response body as "Hello World"
  response.end('Hello World\n');
}).listen(8081);
// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

Node.js – Web Application Creation

1. Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

var http = require('http');

2. The 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:

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


//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
```

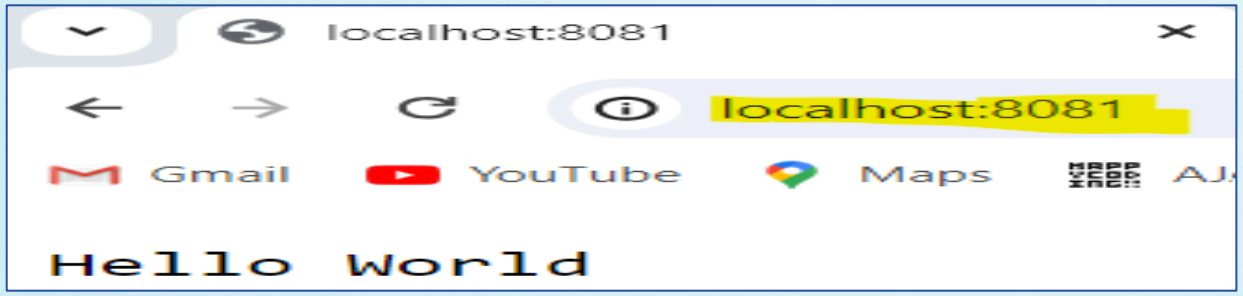
3. The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080

Node.js – Server Start & Stop

- Type the code and save it. The code given in the previous slide is saved as **main.js**.
- Open **Command prompt** and give the command as shown below:

```
Command Prompt - node ma

D:\Node JS Programs>node main.js
Server running at http://127.0.0.1:8081/
```

- Now open the browser and type the url: http://localhost:8081 the browser shows the output of main.js file.



- To stop the server press ^c in the command prompt.

Node.js – Modules

- A set of functions which are to be included in the application. Node.js has a set of **built-in** modules which can be used without any further installation.
- To include a module, use the **require()** function with the name of the module:

var http = require('http');

- Now the application has access to the HTTP module, and is able to create a server:

```
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

Node.js – Core Modules

- The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, the programs need to import the core module first in order to use it in the application.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Node.js – 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:
- The 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:
- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

1. Include the http module

2. Create Server module

3. Content Type

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

Node.js – Timer

Node.js Timer functions are **global functions**. No need to use require() function for this. Timer functions are categorized into set and clear functions. They are:

- **setImmediate()**: It is used to execute immediately.
- **setInterval(function, milliseconds):** it calls the function repeatedly at every millisecond specified.
- **setTimeout(function, milliseconds):** It is used to execute a one-time callback after delay milliseconds.
- **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 – Timer

```
function printName(){
  console.log("VIT University");
}
console.log("Through 'setTimeOut'");
var c1 = setTimeout(printName,2000);
```

Command Prompt

```
D:\Node JS Programs>node TimerPrg1.js
Through 'setTimeOut'
VIT University

D:\Node JS Programs>
```

Node.js – Timer

```
function printName(){
  console.log("VIT University");
}
console.log("through 'setInterval'");
 // calls the function printName in every 2 seconds
var c2 = setInterval(printName,2000);
```

Command Prompt

```
D:\Node JS Programs>node TimerPrg1.js
through 'setInterval'
VIT University
VIT University
VIT University
VIT University
VIT University
VIT University
^C
```

Node.js – Timer

```
function printName(){
  console.log("VIT University");
}
console.log("through 'setInterval'");
 // calls the function printName in every 2 seconds
var c2 = setInterval(printName,2000);
//stops the setInterval-c2
clearInterval(c2);
```

Command Prompt

```
D:\Node JS Programs>node TimerPrg1.js
through 'setInterval'
```

Node.js – Date

The **Date** object works with dates and times. Date objects are created with **new Date().** JavaScript will use the browser's time zone and display a date as a full text string.

E.g.

dateTimeObj = new Date(); //**returns local date and time**

Console.log("Date and time is "+dateTimeObj);

Methods of Date Object:

- getDate() – returns the date value from 1 to 31
- getMonth() – returns the month number from 0 to 11
- getFullYear() – returns the year value in four digits
- getHours() – returns the hours in 24 hr format
- getMinutes() – returns the minutes from 0 to 59
- getSeconds() – returns the seconds from 0 to 59

Node.js – Timer

```
const dateTimeObject = new Date();
const months =['Jan','Feb','Mar','Apr','May','Jun'
  ,'Jul','Aug','Sep','Oct','Nov','Dec'];
console.log("A date-time object is created")
console.log("Date: "+dateTimeObject.toDateString());
console.log("Time: "+dateTimeObject.toTimeString());
let date = dateTimeObject.getDate();
let month = dateTimeObject.getMonth();
let year = dateTimeObject.getFullYear();
let hours = dateTimeObject.getHours();
let minutes = dateTimeObject.getMinutes();
let seconds = dateTimeObject.getSeconds();
console.log("\displaying date(yyyy-mm-dd) and time")
console.log(year + "-" + (month+1) + "-" + date +" "
  + hours + ":" + minutes + ":" + seconds);
console.log("Date in d-mon-yyyy: "+
  date+"-"+months[month]+"-"+year);
```

```
D:\Node JS Programs>node NodeDateObj.js
A date-time object is created
Date: Wed Mar 06 2024
Time: 09:26:40 GMT+0530 (India Standard Time)
displaying date(yyyy-mm-dd) and time
2024-3-6 9:26:40
Date in d-mon-yyyy: 6-Mar-2024
```

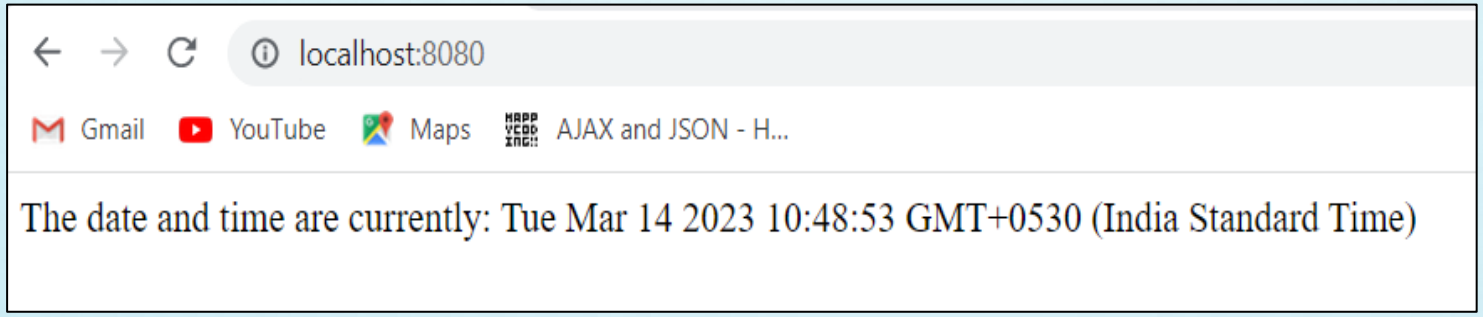
Node.js – User Defined Module

Can create user **defined modules**, and easily include them in the applications. Use the **exports** keyword to make properties and methods available outside the module file. Otherwise, simple require() is enough to include the user defined module.

Save this code in a file called "**myfirstmodule.js**"

```
exports.myDateTime = function () {
  return Date();
};
```

```
var http = require('http');
var dt = require('./myfirstmodule');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("The date and time are currently: " + dt.myDateTime());
  res.end();
}).listen(8080);
```



Node.js – User Defined Module

Save this code in a file called "**FactModule.js**"

```
exports.fact= function(x){
    var f=1;
    for(i=1;i<=x;i++)
        f = f * i;
    return(f);
}
```

```
var httpmod = require('http');
var myMod = require('./FactModule');
httpmod.createServer(function(req,res){
    res.writeHead(200,{'Content-Type':'text/html'});
    res.write("Factorial of 5 is "+myMod.fact(5));
    res.end();
}).listen(3000);
console.log("Server running at port number 3000");
```

Node.js – User Defined Module

Update the "**FactModule.js**" by including the private function 'localFun'

```
exports.fact= function(x){
    var f=1;
    for(i=1;i<=x;i++)
        f = f * i;
    return(f);
}
localFun = function(){
    return("good vibes");
}
```

```
var httpmod = require('http');
var myMod = require('./FactModule');
httpmod.createServer(function(req,res){
    res.writeHead(200,{'Content-Type':'text/html'});
    res.write("Factorial of 5 is "+myMod.fact(5));
    //will create error as it is private function
    res.write("<br>calling local function "+myMod.localFun());
    res.end();
}).listen(3000);
console.log("Server running at port number 3000");
```

```
D:\Node JS Programs\UserModule.js:6
        res.write("<br>calling local function "+myMod.localFun());
                                                ^

TypeError: myMod.localFun is not a function
    at Server.<anonymous> (D:\Node JS Programs\UserModule.js:6:48)
    at Server.emit (node:events:513:28)
```

Node.js – url Module

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

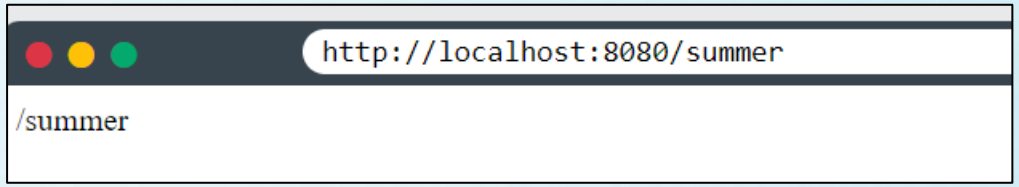
```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

Node.js – Query String

- The function passed into the **http.createServer()** has a req argument that represents the request from the client, as an object (http.IncomingMessage object).
- This object has a property called "**url**" which holds the part of the url that comes after the domain name:

demo_http_url.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```



Node.js – Query String Parsing

- There are built-in modules to easily split the query string into readable parts, such as the URL module.

  url.parse(urlString, slashesDenoteHost).query;

Here,

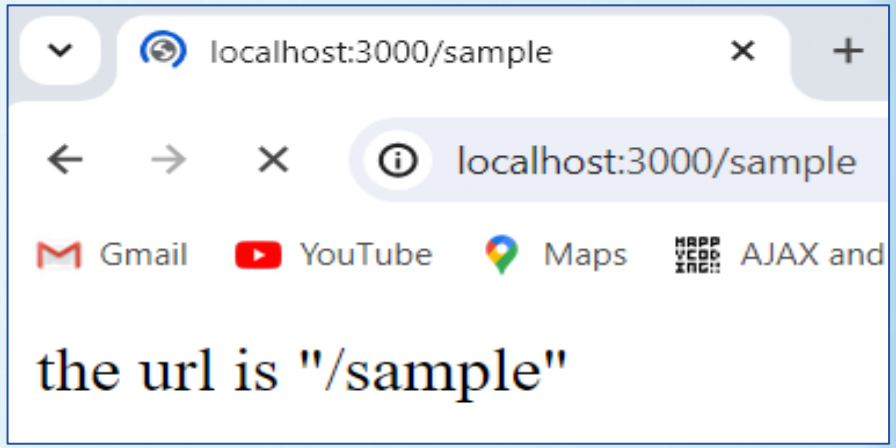
urlString: It holds the URL string which needs to parse.

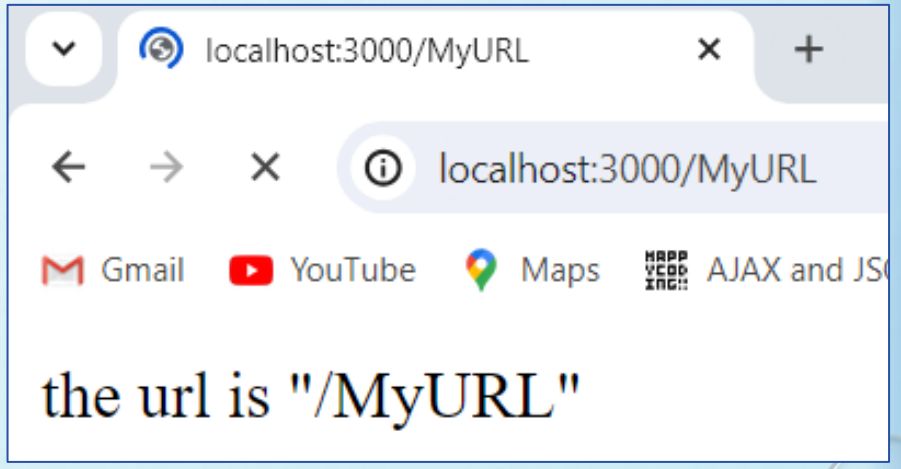
slashesDenoteHost: It is a boolean value. If it set to true then the first token after the literal string // and preceding the next / will be interpreted as the host. For example: //geeksforgeeks.org/web-technology contains the result {host: 'geeksforgeeks.org', pathname: '/web-technology'} rather than {pathname: '//geeksforgeeks.org/web-technology'}. Its default value is false.

Return Value: The url.parse() method returns an object with each part of the address as properties.

Node.js – Query String Parsing

```
var http= require('http')
var url = require('url')
http.createServer(function(req,res){
    res.writeHead(200,{'Content-Type':'text/html'});
    var q = url.parse(req.url,true);
    res.write('the url is "'+req.url+"\"<br>");
    }).listen(3000);
console.log('Server is running in the port 3000');
```





Node.js – File System

- The Node.js file system module allows to work with the file system on the computer. To include the File System module, use the require() method as **require('fs').** Here, the '**fs**' indicates the file system module.
- Common use of the File System module is read, create, update, delete and rename the files.
- The 'fs' module has the following methods for file processing:

1. **fs.open()** – can use to create and append a file.
2. **fs.readFile()** – can read the content of the file if it exists.
3. **fs.writeFile()** – can write a new content if the file not exists otherwise erase the previous contents and write a new content.
4. **fs.appendFile()** – can add the new content at the end of the file
5. **fs.unlink()** – used to delete a file.
6. **fs.rename()** – update the name of an existing file.
7. **fs.copyFile()** – creates a copy of a file

Node.js – File Server

- To make Node.js behave as a file server and serve the file requested by the client.
- Create two html files and save them in the same folder as the node.js files.

summer.html

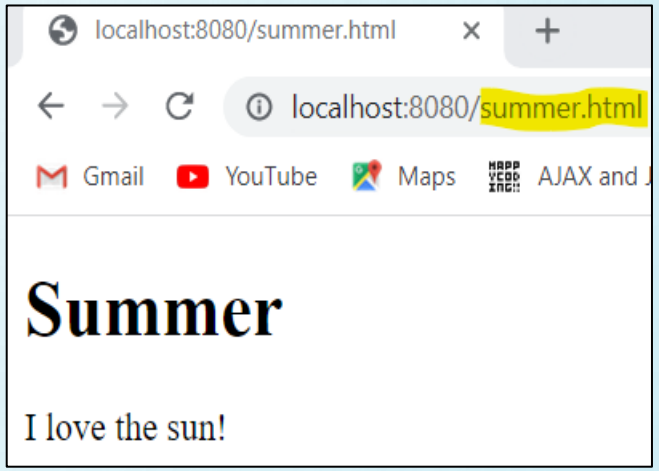
```html
<!DOCTYPE html>
<html>
<body>
<h1>Summer</h1>
<p>I love the sun!</p>
</body>
</html>
```

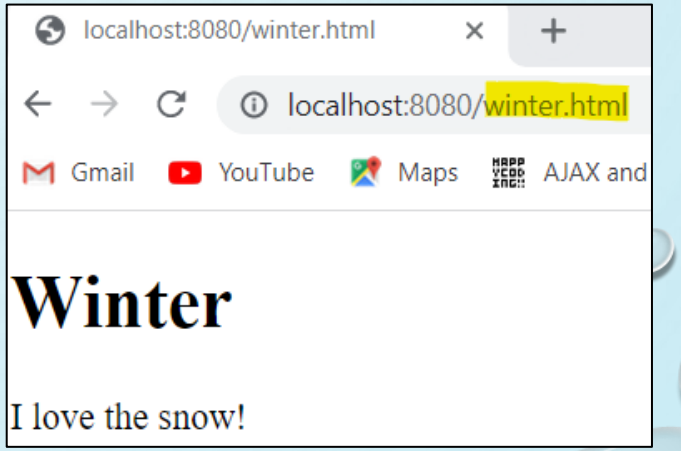
winter.html

```html
<!DOCTYPE html>
<html>
<body>
<h1>Winter</h1>
<p>I love the snow!</p>
</body>
</html>
```

demo_fileserver.js:

```javascript
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```





Node.js – open file

- The **fs.open()** method takes a "flag" as the second argument, if the flag is "**w**" for "**writing**", the specified file is opened for writing. If the file does not exist, an empty file is created:

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

fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});
```

Node.js – writeFile & appendFile

The **fs.writeFile()** method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created.

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

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

The **fs.appendFile()** method adds the specified content at the end of the file if it exists. If the file does not exist, a new file, containing the specified content, will be created.

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

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {
  if (err) throw err;
  console.log('Updated!');
});
```

Node.js – unlink file

To delete a file with the File System module, use the **fs.unlink()** method.

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

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

To rename a file with the File System module, use the **fs.rename()** method.

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

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

Node.js – Node Package Manager

Node Package Manager provides two main functionalities:

- It provides **online repositories** for node.js packages/modules which are searchable on search.nodejs.org
- It also provides **command line utility** to install Node.js packages, do version management and dependency management of Node.js packages.
- NPM is a package manager for Node.js packages, or modules. www.npmjs.com hosts thousands of free packages to download and use. The NPM program is installed on the computer when install Node.js.
- A **package** in Node.js contains all the files needed for a module. **Modules** are JavaScript libraries that can include in the project.

Node.js – NPM

Have download and install package “upper-case”

```
C:\Users\Your Name>npm install upper-case
```

NPM creates a folder named "node_modules", where the package will be placed. All packages that will be installed in the future will be placed in this folder.

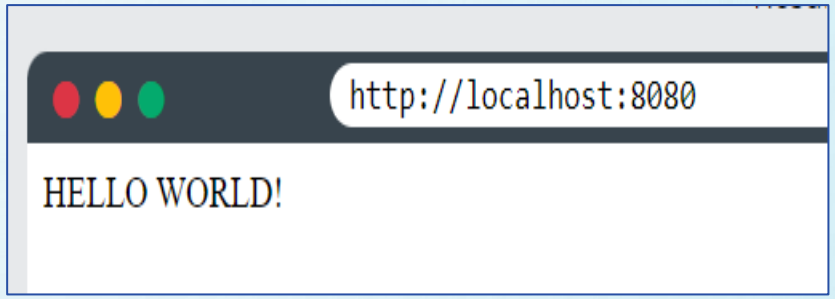
The directory has a folder structure like this:

C:\Users\My Name\node_modules\upper-case

Node.js – NPM

Once the package is installed, it is ready to use. Include the "**upper-case**" package the same way you include any other module:

```
var http = require('http');
var uc = require('upper-case');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc.upperCase("Hello World!"));
  res.end();
}).listen(8080);
```



Node.js – Events Module

- Every action on a computer is an event like when a connection is made or a file is opened. Node.js allows us to create and handle custom events easily by using events module.
- Event module includes EventEmitter class which can be used to raise and handle custom events.
- Objects in Node.js can fire events, like the **readStream** object fires events when opening and closing a file:

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
  console.log('The file is open');
});
```

```
The file is open
```

Node.js – Events Emitter

- Node.js has a built-in module, called "Events", where you can **create-**, **fire-**, and **listen** for- your own events.
- To include the built-in Events module use the require() method. In addition, all event properties and methods are an instance of an **EventEmitter** object. To be able to access these properties and methods, create an EventEmitter object.
- Can assign event handlers to the user defined events with the EventEmitter object.
- The emit() method can be used when fire an event.

Node.js – User Defined Event

- First import the **'events'** module and then create an object of **EventEmitter** class.
- Then specify event handler function using **on()** function. The on() method requires name of the event to handle and callback function which is called when an event is raised.
- The **emit()** function raises the specified event. First parameter is name of the event as a string and then arguments.
- An event can be emitted with zero or more arguments. You can specify any name for a custom event in the emit() function.

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

Command Prompt

```
D:\Node JS Programs>node EventEmitterDemo2.js
First subscriber: This is my first Node.js event emitter example.
```

Node.js – Listener with Event

Can also use **addListener()** methods to subscribe for an event

```
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('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');

// Raising SecondEvent
em.emit('SecondEvent', 'This is my second Node.js event emitter example.');
```

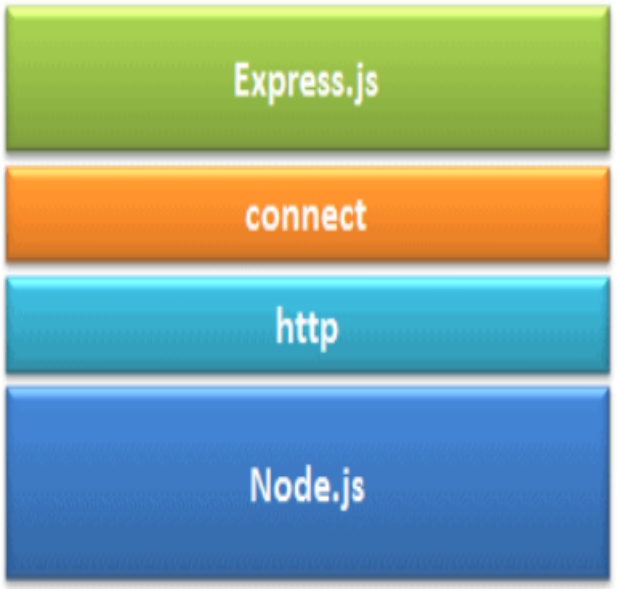
```
D:\Node JS Programs>node EventWithListener1.js
First subscriber: This is my first Node.js event emitter example.
First subscriber: This is my second Node.js event emitter example.
```

Node.js – Express JS Introduction

- Express is a fast, assertive, essential and moderate **web framework** of **Node.js**. You can assume express as a layer built on the top of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.

  Features:

  - It can be used to design single-page, multi-page and hybrid web applications.
  - It allows to setup middlewares to respond to HTTP Requests.
  - It defines a routing table which is used to perform different actions based on HTTP method and URL.
  - It allows to dynamically render HTML Pages based on passing arguments to templates.



Express.js

Node.js – Express JS Installation

You can **install express.js** using **npm**. The following command will install latest version of express.js **globally** on your machine so that every Node.js application on your machine can use it.

```
Command Prompt

D:\Node JS Programs>npm install -g express

changed 57 packages in 2s

7 packages are looking for funding
  run `npm fund` for details

D:\Node JS Programs>
```

To know the installed version use this command

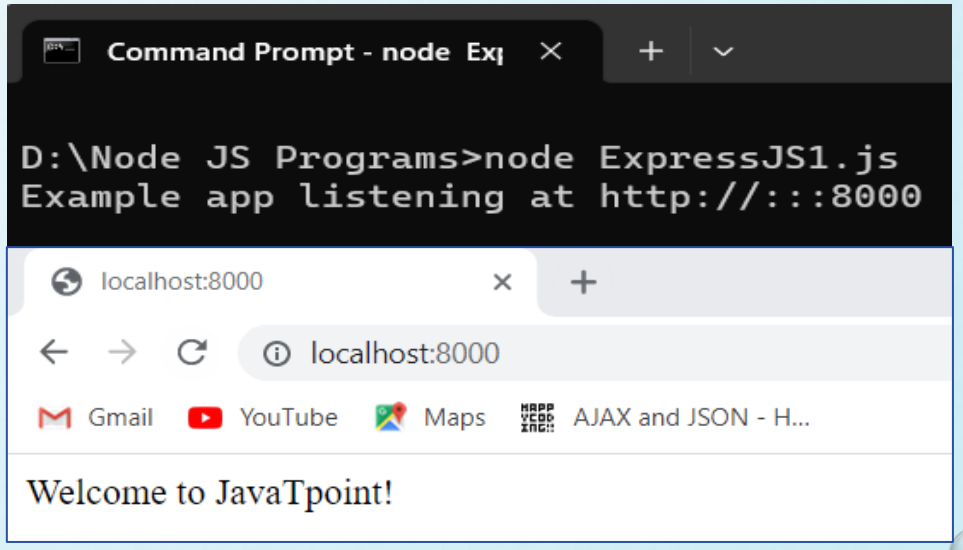
```
Command Prompt

D:\Node JS Programs>npm ls -g installed express
C:\Users\avran\AppData\Roaming\npm
`-- express@4.18.2
```

Eventhough the express installed globally, it has to be copied in the project folder. Otherwise install locally like **'>npm install express'**

Node.js – Express Sample Code

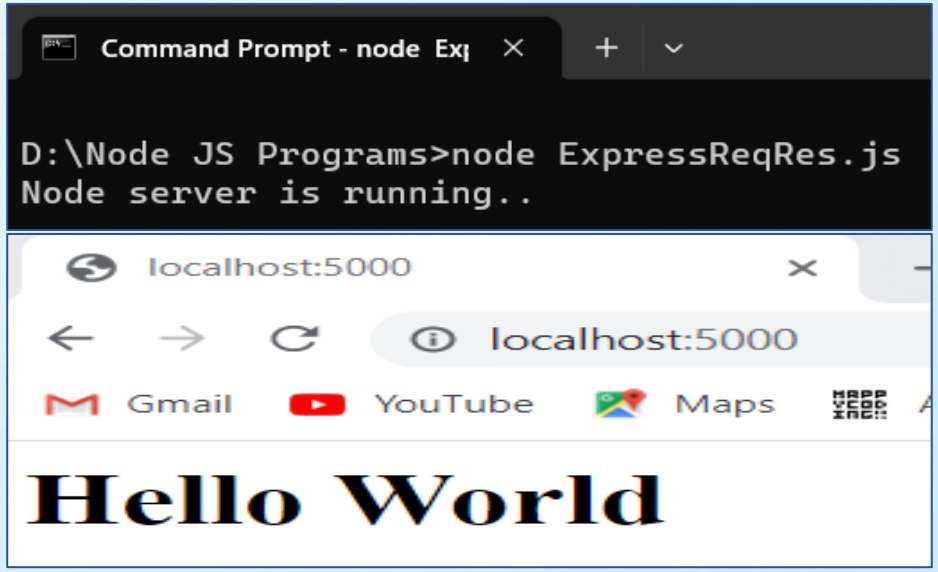
Save this code as ExpressJS1.js and run as shown here:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Welcome to JavaTpoint!');
});
var server = app.listen(8000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Example app listening at http://%s:%s', host, port);
});
```



Node.js – Express Sample Code

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

app.get('/', function (req, res) {
    res.send('<html><body><h1>Hello World</h1></body></html>');
});

app.post('/submit-data', function (req, res) {
    res.send('POST Request');
});

app.put('/update-data', function (req, res) {
    res.send('PUT Request');
});

app.delete('/delete-data', function (req, res) {
    res.send('DELETE Request');
});

var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```



Node.js – Request Objects

Express.js Request and Response objects are the parameters of the callback function which is used in Express applications.

The express.js request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

Properties	Description
req.app	This is used to hold a reference to the instance of the express application that is using the middleware.
req.baseurl	It specifies the URL path on which a router instance was mounted.
req.body	It contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as body-parser.
req.cookies	When we use cookie-parser middleware, this property is an object that contains cookies sent by the request.
req.hostname	It contains the hostname from the "host" http header.
req.ip	It specifies the remote IP address of the request.
req.params	An object containing properties mapped to the named route ?parameters?. For example, if you have the route /user/:name, then the "name" property is available as req.params.name. This object defaults to {}.
req.path	It contains the path part of the request URL.
req.query	An object containing a property for each query string parameter in the route.
req.route	The currently-matched route, a string.
req.signedcookies	When using cookie-parser middleware, this property contains signed cookies sent by the request, unsigned and ready for use.

Node.js – Request Object Methods

Method	Description	Example
req.accepts (types)	used to check whether the specified content types are acceptable, based on the request's Accept HTTP header field.	req.accepts('html'); //=>?html? req.accepts('text/html'); // => ?text/html?

req.get(field)	returns the specified HTTP request header field.	req.get('Content-Type'); // => "text/plain"
		req.get('content-type'); // => "text/plain"
		req.get('Something'); // => undefined

req.is(type)	returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter.	// With Content-Type: text/html; charset=utf-8
		req.is('html');
		req.is('text/html');
		req.is('text/*');
		// => true

req.param(name [, defaultValue])	used to fetch the value of param name when present.	// ?name=sasha
		req.param('name') // => "sasha"
		// POST name=sasha
		req.param('name') // => "sasha"
		// /user/sasha for /user/:name
		req.param('name') // => "sasha"

Node.js – Response Objects

- The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.

  It sends response back to the client browser.

  It facilitates you to put new cookies value and that will write to the client browser.

  Once you res.send() or res.redirect() or res.render(), you cannot do it again, otherwise, there will be uncaught error.

Properties	Description
res.app	It holds a reference to the instance of the express application that is using the middleware.
res.headersSent	It is a Boolean property that indicates if the app sent HTTP headers for the response.
res.locals	It specifies an object that contains response local variables scoped to the request

Node.js – Response Object Methods

Method	Description	Example
res.append(field[,value])	appends the specified value to the HTTP response header field. That means if the specified value is not appropriate then this method redress that.	res.append('Link', ['<http://localhost/>', '<http://localhost:3000/>']); res.append('Warning', '199 Miscellaneous warning');
res.attachment([filename])	Used to send a file as an attachment in the HTTP response.	res.attachment('path/to/js_pic.png');

res.cookie(name, value [, options])	is used to set a cookie name to value. The value can be a string or object converted to JSON.	res.cookie('name', 'Aryan', { domain: '.xyz.com', path: '/admin', secure: **true** });
		res.cookie('Section', { Names: [Aryan,Sushil,Priyanka] });
		res.cookie('Cart', { items: [1,2,3] }, { maxAge: 900000 });

res.clearCookie(name [, options])	used to clear the cookie specified by name.	res.clearCookie('name', { path: '/admin' });
res.end([data] [, encoding])	used to end the response process.	res.end(); res.status(404).end();

Node.js – Response Object Methods

Method	Description	Example
res.get(field)	provides HTTP response header specified by field.	res.get('Content-Type');
res.json([body])	returns the response in JSON format.	res.json(null) res.json({ name: 'ajeet' })

res.send([body])	used to send HTTP response.	res.send(new Buffer('whoop'));
		res.send({ some: 'json' });
		res.send('
		.....some html
		');

res.set(field [, value])	used to set the response of HTTP header field to value	res.set('Content-Type', 'text/plain');
		
		res.set({
		'Content-Type': 'text/plain',
		'Content-Length': '123',
		})

Node.js – Response Object Methods

The methods on the response object (res) in the following table can send a response to the client, and terminate the request-response cycle. If none of these methods are called from a route handler, the client request will be left hanging.

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

Node.js – GET Method

GET and **POST** both are two common HTTP requests used for building REST API's. GET requests are used to send only **limited amount of data** because data is sent into header.

- Save this as "**FormInput1.html**"
- Open this code in the browser by right click over the file. The url will be
- **File:///d:/node js programs/FormInput1.html**

```
<html>
<body>
<form action="http://127.0.0.1:8000/Process_Get1" method="GET">
First Name: <input type="text" name="first_name">  <br>
Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Node.js – GET Method Sample Program1

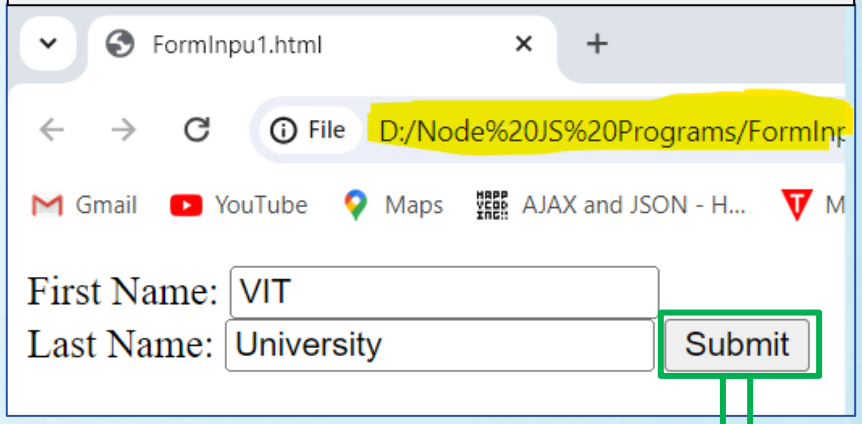
Save this as "**Process_Get1.js**". Open command prompt and type
>node Process_Get1.js

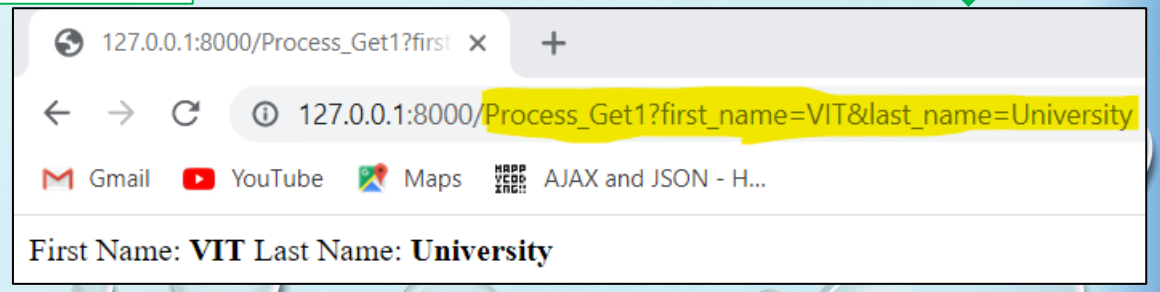
```
//importing express framework
var express = require('express');
var app = express();
//action for Get method
app.get('/Process_Get1', function (req, res) {
//receiving input through form
   var opStr = "First Name:<b> "+req.query['first_name']+
               "</b> Last Name: <b>"+req.query['last_name']+"</b>";
   console.log(opStr);  //printing in the console
   res.send(opStr);  //displays in the browser
}
)
var server = app.listen(8000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
}
)
```

Server side console output

```
Command Prompt - node Pro

D:\Node JS Programs>node Process_Get1.js
Example app listening at http://:::8000
```

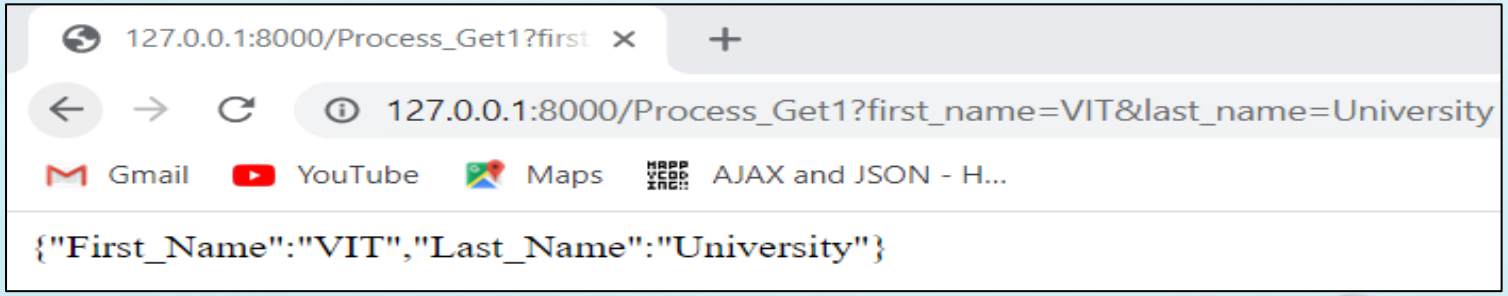
Browser Output





Node.js – GET Method with JSON data

```
//importing express framework
var express = require('express');
var app = express();
//action for Get method
app.get('/Process_Get1', function (req, res) {
//receiving input through form
    var JSONObj = {First_Name:req.query['first_name'],
                   Last_Name:req.query['last_name']
    };
    console.log(JSONObj);  //printing in the console
    res.send(JSON.stringify(JSONObj));  //displays in the browser
}
)
var server = app.listen(8000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
}
)
```



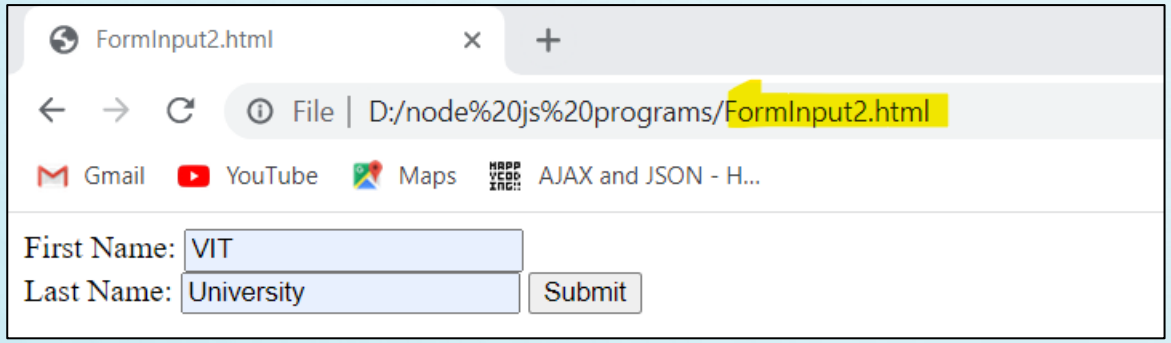
Node.js – POST Method

- **POST** requests are used to send **large amount of data.**
- Express.js facilitates you to handle GET and POST requests using the instance of express.
- Post method facilitates you to send large amount of data because **data** is send in the **body**.
- Post method is secure because data is not visible in URL bar

Node.js – POST Method Sample Code

FormInput2.html

```
<html>
<body>
<form action="http://127.0.0.1:8000/Process_Post1" method="POST">
First Name: <input type="text" name="first_name">  <br>
Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```



Node.js – POST Method Sample Code

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
// Create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })

app.post('/Process_Post1', urlencodedParser, function (req, res) {
    // Prepare output in JSON format
    response = {
        first_name:req.body.first_name,
        last_name:req.body.last_name
    };
    console.log(response);
    res.end(JSON.stringify(response));
});
var server = app.listen(8000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

Command Prompt - node Pr

```
D:\Node JS Programs>node Process_Post1.js
Example app listening at http://:::8000
{ first_name: 'VIT', last_name: 'University' }
```

127.0.0.1:8000/Process_Post1

127.0.0.1:8000/Process_Post1

Gmail YouTube Maps AJAX and JSON - H...

{"first_name":"VIT","last_name":"University"}

Node.js – Routing

- Routing is made from the word route. It is used to determine the **specific behavior of an application**. It specifies how an application responds to a client request to a particular route, URI or path and a specific HTTP request method (GET, POST, etc.). It can handle different types of HTTP requests.

Each route can have one or more handler functions, which are executed when the route is matched. Route definition takes the following structure:

```
app.METHOD(PATH, HANDLER)
```

Here,

app is an instance of express.

METHOD is an HTTP request method, in lowercase.

PATH is a path on the server.

HANDLER is the function executed when the route is matched.

Node.js – Simple Routes

```
const express = require('express')
const app = express()
```

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})
```

```
app.post('/', (req, res) => {
  res.send('Got a POST request')
})
```

```
app.put('/user', (req, res) => {
  res.send('Got a PUT request at /user')
})
```

```
app.delete('/user', (req, res) => {
  res.send('Got a DELETE request at /user')
})
```

Node.js – Routing Sample Program

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
   console.log("Got a GET request for the homepage");
   res.send('Welcome to JavaTpoint!');
})
app.post('/', function (req, res) {
   console.log("Got a POST request for the homepage");
   res.send('I am Impossible! ');
})
app.delete('/del_student', function (req, res) {
   console.log("Got a DELETE request for /del_student");
   res.send('I am Deleted!');
})
app.get('/enrolled_student', function (req, res) {
   console.log("Got a GET request for /enrolled_student");
   res.send('I am an enrolled student.');
})
// This responds a GET request for abcd, abxcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
   console.log("Got a GET request for /ab*cd");
   res.send('Pattern Matched.');
})
var server = app.listen(8000, function () {
var host = server.address().address
  var port = server.address().port
console.log("Example app listening at http://%s:%s", host, port)
})
```

Node.js – Routing Sample Program Output

Command Prompt - node Ro

```
D:\Node JS Programs>node Routing1.js
Example app listening at http://:::8000
Got a GET request for the homepage
```

localhost:8000

Welcome to JavaTpoint!

Command Prompt - node Ro

```
D:\Node JS Programs>node Routing1.js
Example app listening at http://:::8000
Got a GET request for the homepage
Got a GET request for the homepage
Got a GET request for /enrolled_student
```

localhost:8000/enrolled_student

I am an enrolled student.

Command Prompt - node Ro

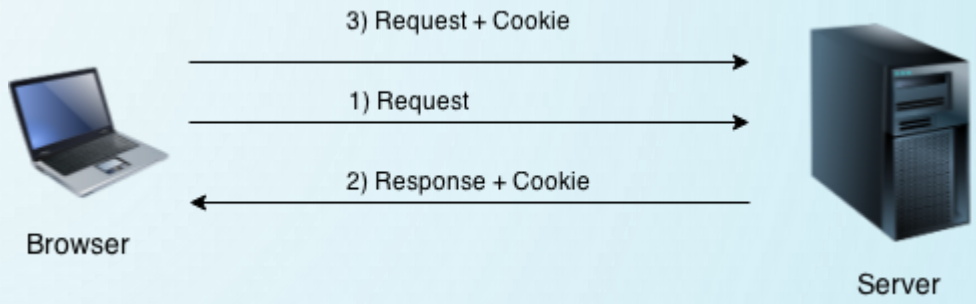
```
D:\Node JS Programs>node Routing1.js
Example app listening at http://:::8000
Got a GET request for the homepage
Got a GET request for the homepage
Got a GET request for /enrolled_student
Got a GET request for /ab*cd
```

localhost:8000/ablkcd

Pattern Matched.

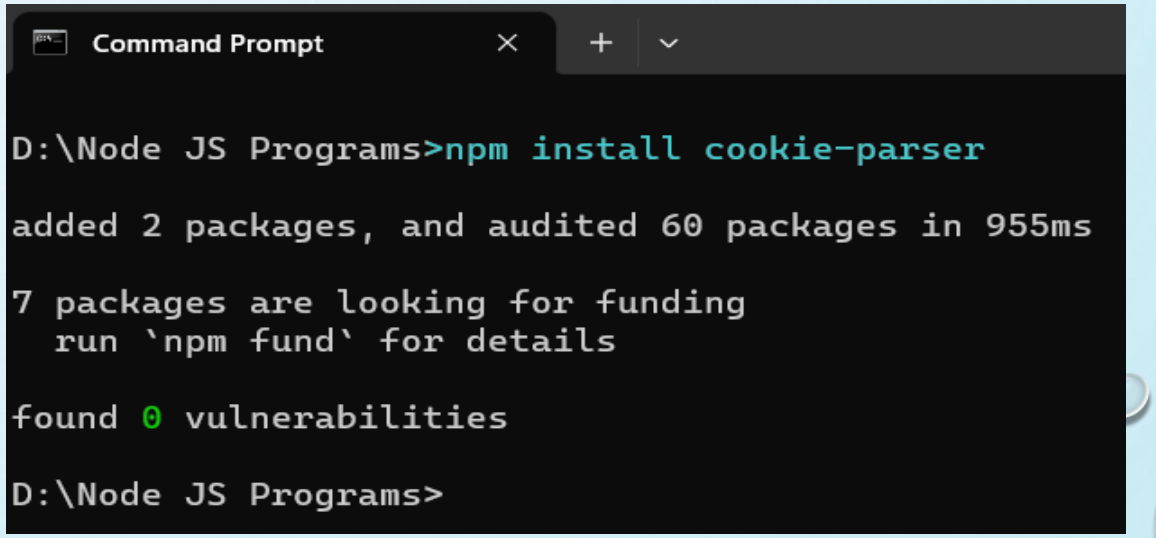
Node.js – Cookie

Cookies are small piece of information i.e. sent from a website and stored in **user's web browser** when user browses that website. Every time the user loads that website back, the browser sends that stored data back to website or server, to recognize user.



install **cookie-parser** middleware through **npm** by using this command:

> npm install cookie-parser



Node.js – Cookie

To use cookies with Express, we will require the **cookie-parser**. cookie-parser is a middleware which *parses cookies attached to the client request object*. To use it, we will require it in the code file; this can be used the same way as we use other middleware. Here, we will use the following code.

```
var cookieParser = require('cookie-parser');
app.use(cookieParser());
```

To create a new cookie use this command:

res.cookie({cookie1Name:'Cookie1 Value' [,cookie2Name:'Cookie2 Value',...})

Sample:

```
app.get('/', function(req,res){
   res.cookie('myCookie','Web Technologies');
})
```

Node.js – Cookie

Cookie-parser parses Cookie header and populates **req.cookies** with an object keyed by the cookie names. To ***set a new cookie***, let us define a new route in your Express app like –

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

app.get('/', function(req, res){
   res.cookie('name', 'express').send('cookie set'); //Sets name = express
});
```

To add a cookie that **expires**, just pass an object with property 'expire' set to the time when you want it to expire.

res.cookie('name','value',{expire:time in milliseconds +currentTime}

Sample code:

```
//Expires after 360000 ms from the time it is set.
res.cookie(name, 'value', {expire: 360000 + Date.now()});
```

With 'maxAge'

```
//This cookie also expires after 360000 ms from the time it is set.
res.cookie(name, 'value', {maxAge: 360000});
```

Node.js – Cookie

To **delete a cookie**, use the **clearCookie** function. For example, if you need to clear a cookie named **foo**, use the following code.

```
res.clearCookie('foo');
```

Node.js – Session

- HTTP is **stateless**; in order to associate a request to any other request, you need a way to **store user data between HTTP requests**.
- Cookies and URL parameters are both suitable ways to transport data between the client and the server.
- But they are both readable and on the client side. Sessions solve exactly this problem.
- You assign the client an ID and it makes all further requests using that ID. Information associated with the client is stored on the server linked to this ID.

To install it using the following code:

```
>npm install express-session
```

To initialize the session:

```
app.use(session({secret: 'Your_Secret_Key',
resave: true, saveUninitialized: true}));
```

To create a session:

```
req.session.key = value;
```

To access a session

```
res.send(req.session.key);
```

To check for an existing session, use the session name in the control statement, like

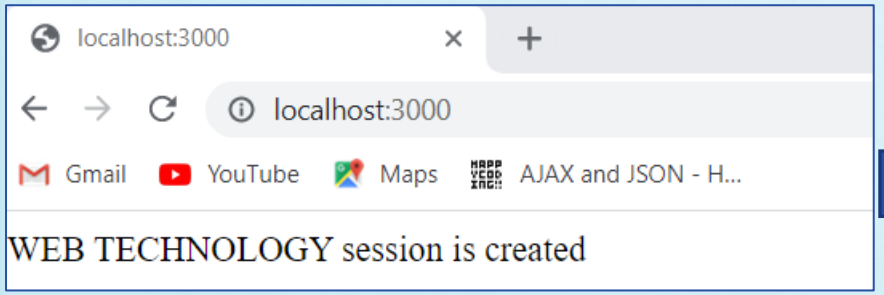
```
if (req.session.key)
```

To remove a session variable use,

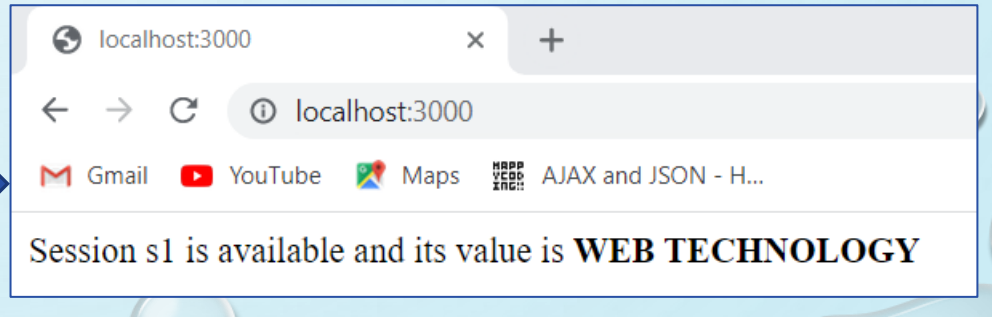
```
delete req.session.key
```

Node.js – Session Sample Program1

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');
var app = express();

app.use(cookieParser());
app.use(session({secret: 'Your_Secret_Key', resave: true, saveUninitialized: true}));
app.get('/', function(req, res){
   if(req.session.s1) //checks the session s1 is available
      res.send("Session s1 is available and its value is <b> "+ req.session.s1+"</b>");
   else {//s1 is not available and it is created
       req.session.s1="WEB TECHNOLOGY";
        res.send(req.session.s1+" session is created");
   }
});
app.listen(3000);
```

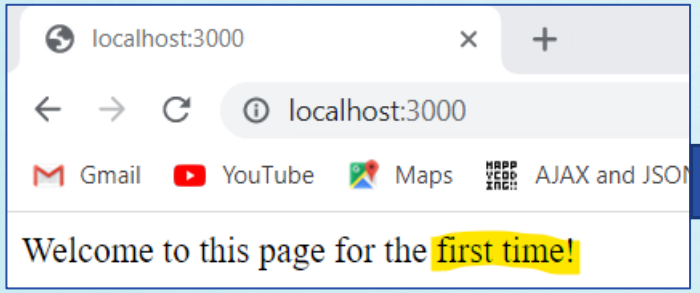


After Page Refresh

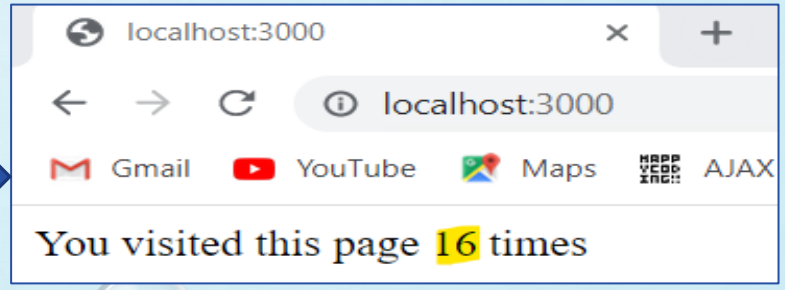


Node.js – Session Sample Program2

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');
var app = express();

app.use(cookieParser());
app.use(session({secret: 'Your_Secret_Key', resave: true, saveUninitialized: true}));
app.get('/', function(req, res){
    if(req.session.page_views){
        req.session.page_views++;
        res.send("You visited this page " + req.session.page_views + " times");
    } else {
        req.session.page_views = 1;
        res.send("Welcome to this page for the first time!");
    }
});
app.listen(3000);
```



After 16th Refresh



Node.js – File Upload

Formidable is a module which helps to **upload** file in Node.js. This can be installed as

>npm install formidable

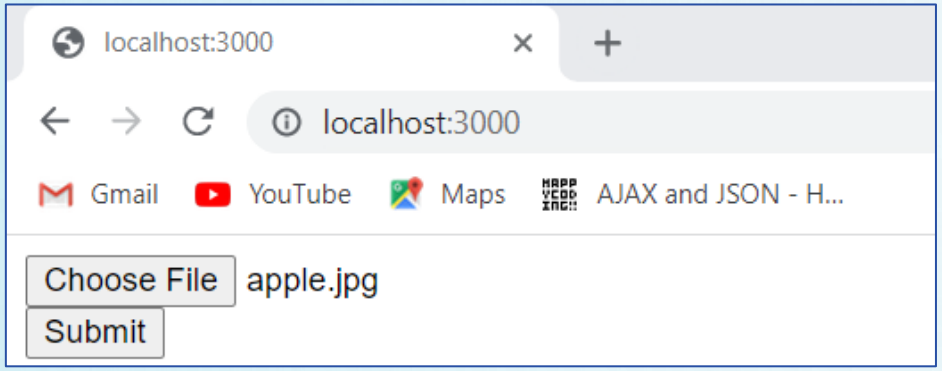
And this can be included in the application. File upload has three process:

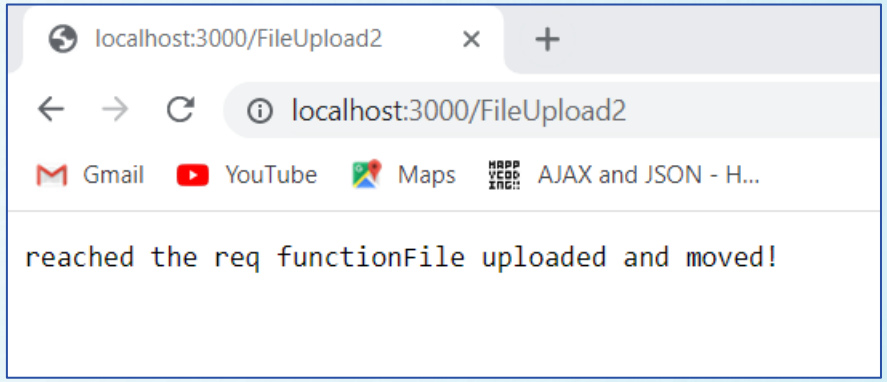
1. Choose the file for uploading
2. File parser should be applied to segregate the fields and file.
3. Copy or move the file from source path to the specified path.

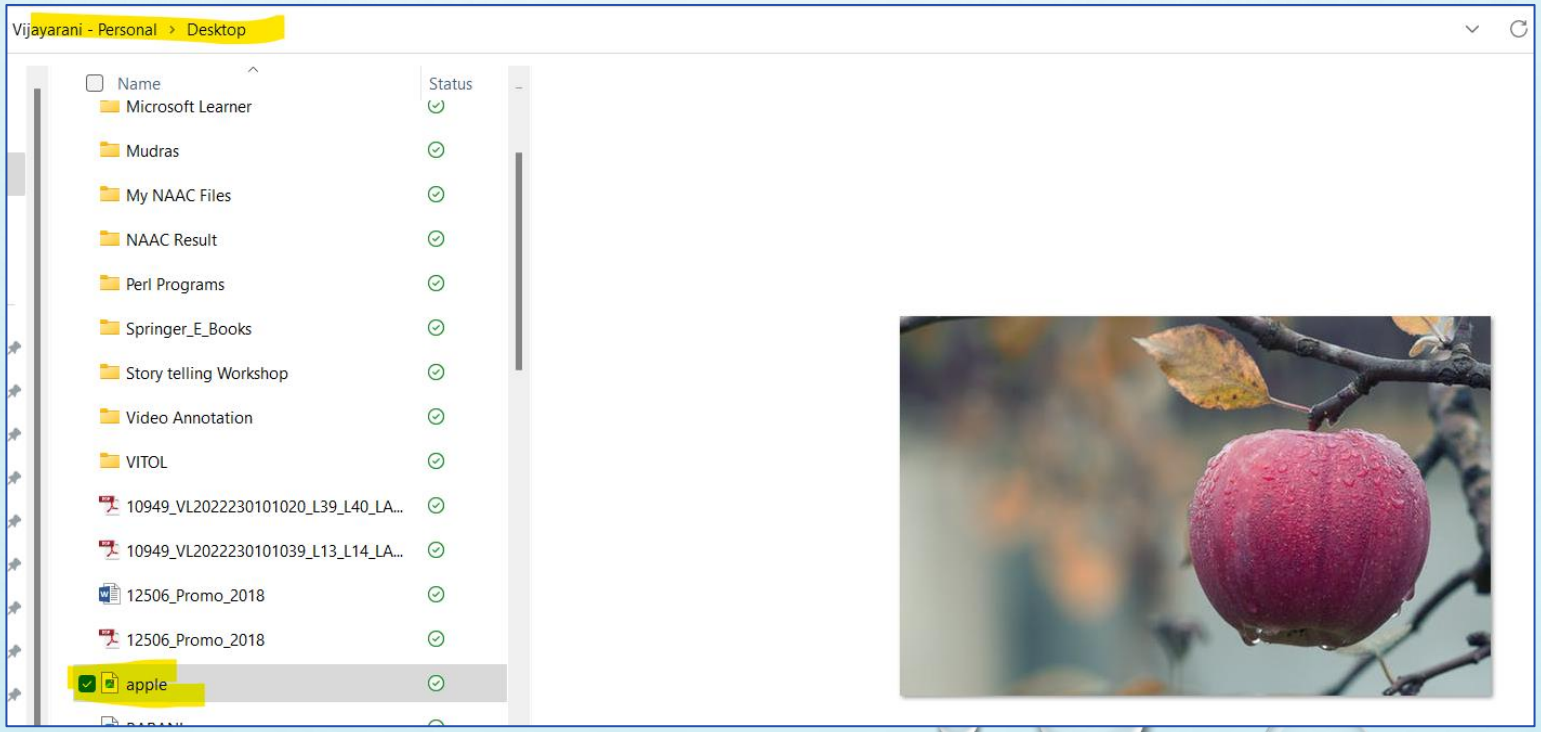
Node.js – File Upload Sample Program

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

http.createServer(function (req, res) {
  if (req.url == '/FileUpload2') {//node js part to upload the selected file
      res.write("reached the req function");
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      var oldpath = files.filetoupload.filepath;
      var newpath = 'C:/Users/avran/OneDrive/Desktop/' + files.filetoupload.originalFilename;
        fs.copyFile(oldpath,newpath,function(err){
         if (err) throw err;
        res.write('File uploaded and moved!');
        res.end();        });
 });
  } else { // HTML part to choose a file for uploading
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="FileUpload2" 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(3000);
```

Node.js – File Upload Sample Program







Node.js – References

1. https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm
2. https://www.w3schools.com/nodejs/
3. https://www.javatpoint.com/nodejs-tutorial
4. https://www.tutorialsteacher.com/nodejs
5. https://www.javatpoint.com/expressjs-tutorial