5.1 Concepts, Working and Features

- ✓ Node.js is an open-source server side runtime environment built on Chrome's V8 JavaScript engine.
- ✓ It provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side application using JavaScript.
- ✓ Node.js was written and introduced by **Ryan Dahl** in 2009.
- ✓ A Node.js app runs in a single process, without creating a new thread for every request.
- ✓ Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking.
- ✓ When Node.js performs an I/O operation, like reading from the network, accessing a database or the file system, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.
- ✓ 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 also provides a rich library of various JavaScript modules to simplify the development of command line application, web application, real-time chat application, REST API server etc.
- ✓ **Node.js = Runtime Environment + JavaScript Library**

➢ What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can add, delete, modify data in your database.
- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

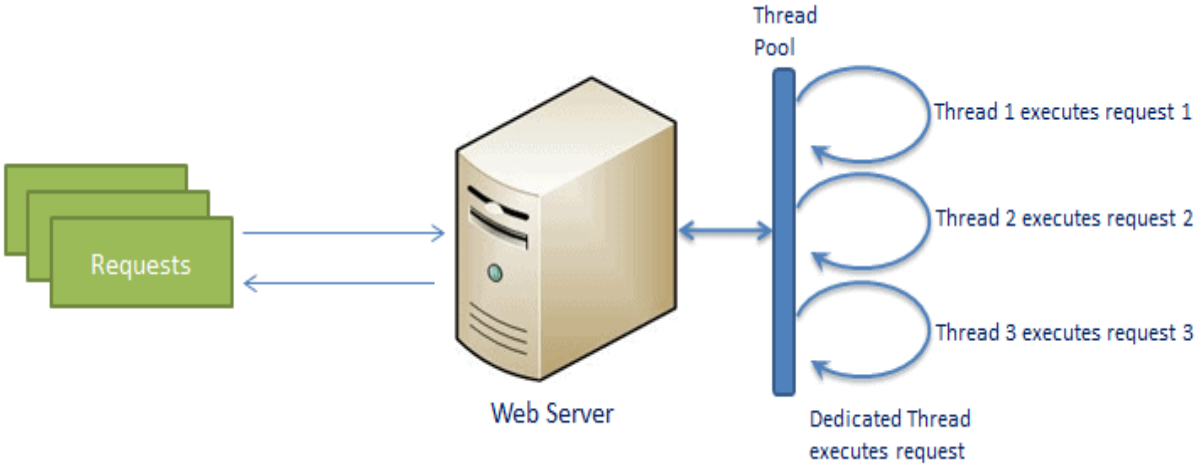
➢ Node.js Process Model

☞Traditional Web Server Model

- In the traditional web server model, each request is handled by a dedicated thread from the thread pool.
- 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.

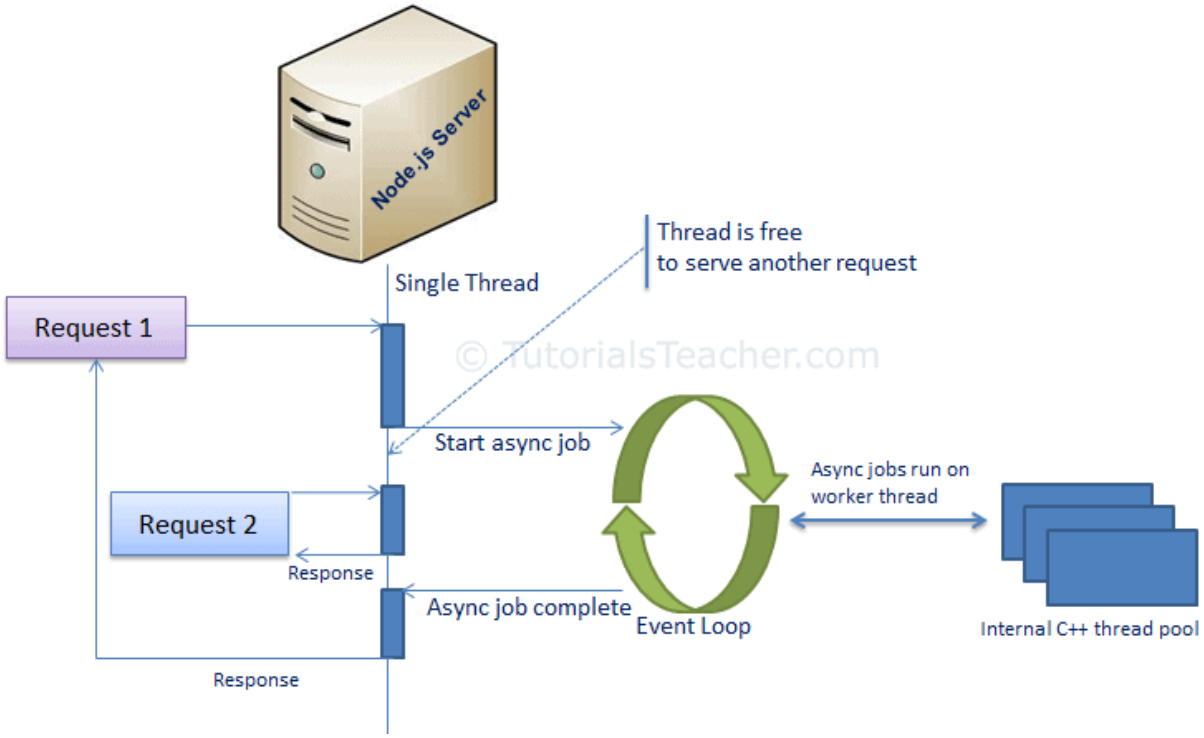
Traditional Webserver Model



☞ Node.js Process Model

- Node.js runs in a single process and the application code runs in a single thread.

Node.js Process Model



- All the user requests to web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request.
- When asynchronous I/O work completes then it processes the request further and sends the response.
- An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes.
- Node.js process model increases the performance and scalability.

➢ Features of Node.js

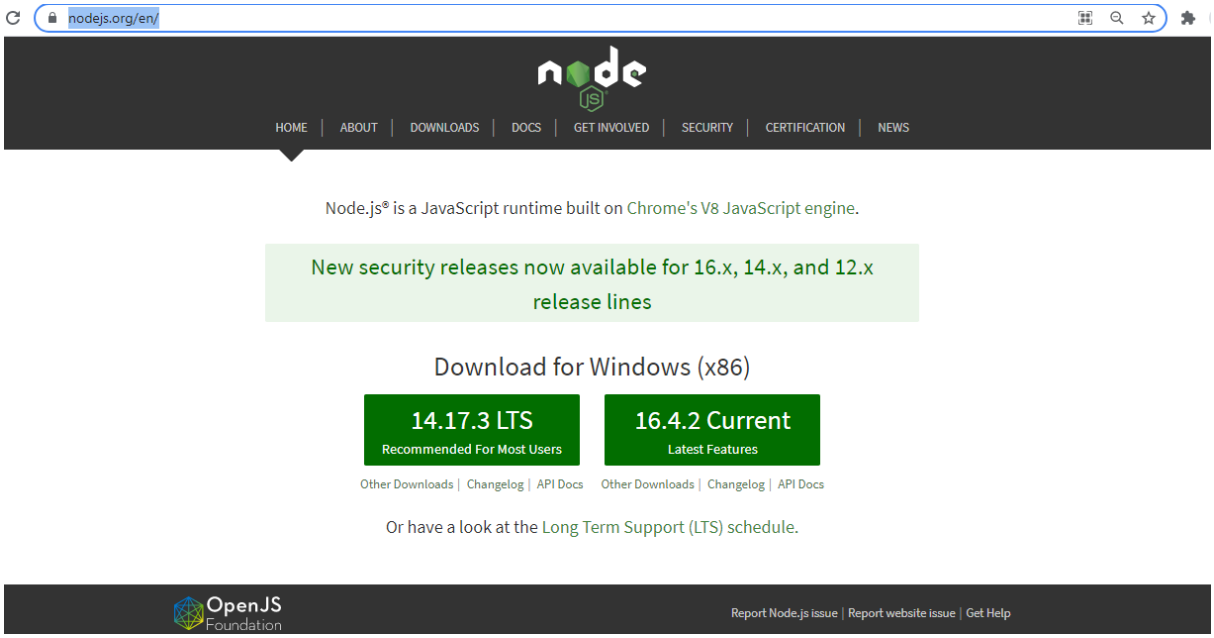
- ☞ **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
- ☞ **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. 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 (Massachusetts Institute of Technology) license.

➢ Where to Use Node.js (Applications)?

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

5.1.1 Downloading Node.js

- we can download the latest version of Node.js installable archive file from https://nodejs.org/en/
- Here, download and installation of node-v14.17.3 LTS.

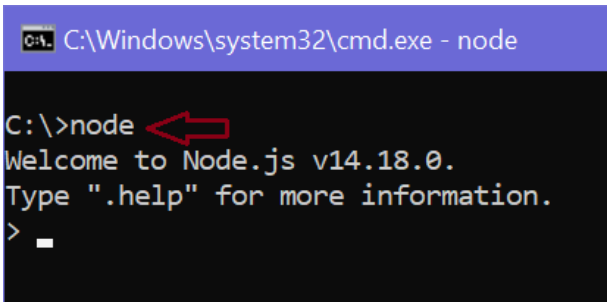


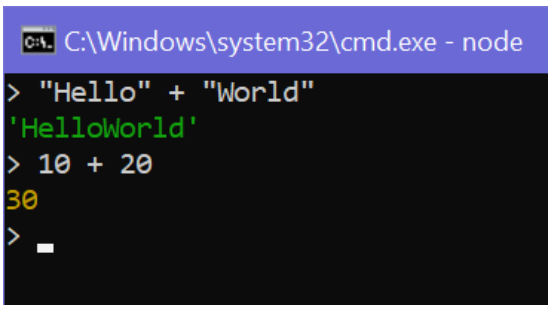
5.2 Setting up Node.js Server (HTTP Server)

5.2.1 Installing on Windows

❖ Node.js Console/REPL

- ✓ Node.js comes with virtual environment called REPL.
- ✓ REPL stands for **Read-Eval-Print-Loop**. It is a quick and easy way to test simple Node.js JavaScript code.
- ✓ we can now test any Node.js/JavaScript expression in REPL. E.g. 10 + 20 will display 30 immediately in new line.



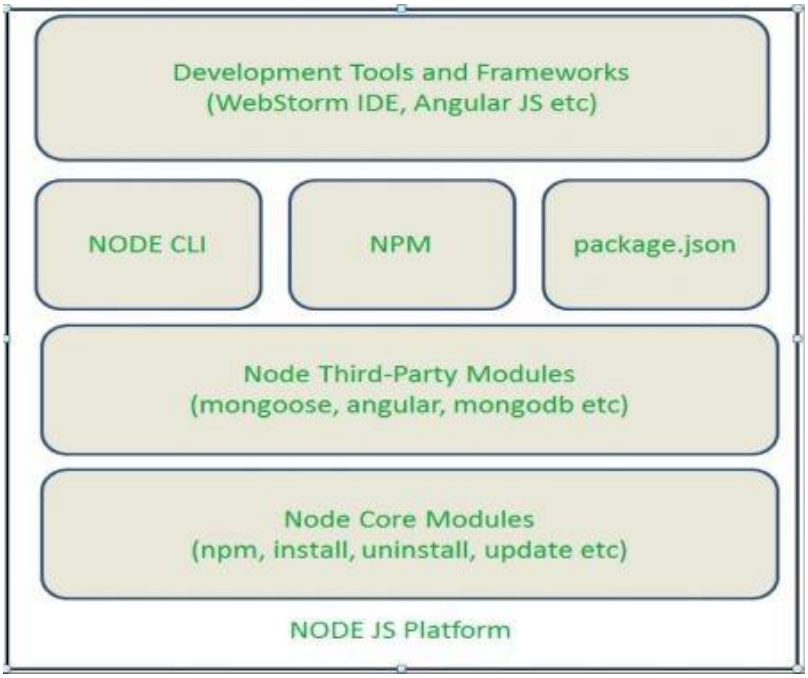


- ✓ The + operator also concatenates strings as in browser's JavaScript.
- ✓ You can also define variables and perform some operation on them.
- ✓ The following table lists important REPL commands.

REPL Command	Description
.help	Display help on all the commands
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
.save filename	Save current Node REPL session to a file.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression.

5.2.2 Node .JS Components

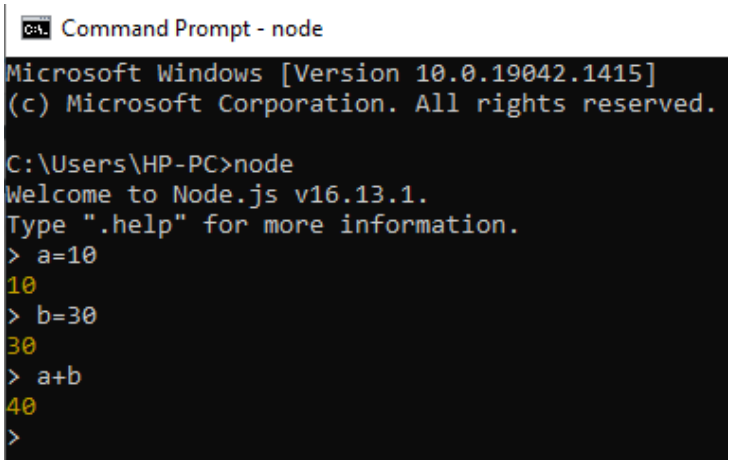
1. Node CLI.
2. NPM.
3. Package .json
4. Node Modules.
5. Development Tools and Frameworks.



1. Node CLI

- ✓ Node JS Platform has a CLI (Command Line Interface) to run basic commands and also script files.
- ✓ When we install Node JS Platform, by default we will get this component. We do not need to any extra configurations for this component.
- ✓ To access **Node CLI** open Command prompt and type **"Node"** Command.
- ✓ Now we are able to see NODE CLI i.e. **">",** that means our Node JS Setup is working fine.

- ✓ Here we can run basic Java Script commands one by one.



2. NPM (Node Package Manager)

- ✓ NPM is used to install, update, uninstall and configure Node JS Platform modules/packages very easily.
- ✓ When we install Node JS Base Platform, it installs only few components, modules and libraries like Node CLI, NPM etc.
- ✓ By default, Node JS platform install NPM module.
- ✓ When we install any other required modules, means for each module or package, Node JS maintains a separate folder here.
- ✓ To check npm version, run "**npm -v** command.

3. package.json

- ✓ **"package.json"** is plain text file in JSON format. It is used to manage our application required module dependencies. We should place this file in our application root folder.
- ✓ It defines information like our application name, module dependencies, module versions etc.
- ✓ This configurations file is very important and requires more time to explain in detail.
- ✓ **Sample :** package.json file;

```
{
    "name" : "sampleapp",
    "version" : "1.0.0",
    "dependencies" : {
    }
}
```

4. Node Modules

- ✓ Node JS is more modular platform. Each functionality is implemented by a separate module or package.

- ✓ It has some core modules like npm, install, uninstall, update etc and rest all modules are third-party modules.
- ✓ When we install Node JS Platform, by default only one module is installed i.e. **npm** module. We need to use "npm" command to install required modules one by one.
- ✓ Node JS has thousands of modules, but here we are going to use some of the popular modules.

5. Development Tools and Frameworks

- ✓ As Node JS Platform became very popular to develop Data-Sensitive Real-time and Network applications, many companies have developed some tools and framework to ease and reduce the overhead of Node JS applications.

Category	Framework/Tools
IDE	Eclipse with node.js plugins, Jetbrains ,Webstorm, Cloud9 IDE, Visual Studio Node JS Toolkit
Database	MongoDB
UI Build Tools	Grunt, Yeoman, Gulp
CLI	Node CLI, grunt-cli
Authentication	Passport.js
UI Frameworks	Backbone.js, Angular.js, Ember.js
Template Engine	Jade, EJS, Hogan.JS
CSS Engine	Stylus, LESS, Compass
Unit Testing Frameworks	Jasmin, Node Unit

5.2.2.1 Node.js Modules

- Node.Js modules to be the same as JavaScript libraries.
- Node.js treats each JavaScript file as a separate module.
- Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.
- Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope.
- Also, each module can be placed in a separate **.js** file under a separate folder.

❖ Node.js Module Types: Node.js includes three types of modules:

1. **Core Modules (Built-in Modules)**
2. **Local Modules**
3. **Third Party Modules**

1. Node.js Core Modules

- ✓ Node.js is a light weight framework. The core modules include minimum functionalities of Node.js.
- ✓ These core modules are load automatically when Node.js process starts. So that, we need to **import** the core module first in order to use it in our application.
- ✓ The following table lists some of the **important core modules** in Node.js.

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.

➢ Loading Core Modules

- In order to use Node.js core or **NPM modules**, first need **to import it using require() function** as shown below.
- **Syntax : var module = require('module_name');**
- The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.
- **Example:** Load and Use Core http Module

```
var http = require('http');
var server = http.createServer(function(req, res){
      //write code here
});
server.listen(5000);
```

- In the above example, **require()** function **returns an object** because http module returns its functionality as an object, we can then use its properties and methods using **dot**(**.**) notation e.g. http.createServer().

2. Node.js Local Module

- ✓ Local modules are modules created locally in Node.js application. These modules include different functionalities of application in separate files and folders.
- ✓ For example, if we need to connect to MongoDB and fetch data then we can create a module for it, which can be reused in our application.
- ✓ **Example:** Let's write simple logging module which logs the information, warning or error to the console.
- ✓ In Node.js, module should be placed in a separate JavaScript file. So, create a **Log.js** file and write the following code in it.

Log.js

```
var log = {
        info: function (info) {
           console.log('Info: ' + info);
        },
        warning:function (warning) {
           console.log('Warning: ' + warning);
        },
        error:function (error) {
           console.log('Error: ' + error);
        }
    };
module.exports = log;
```

- In the above example of logging module, we have created an object with three functions - **info(), warning() and error().** At the end, we have assigned this object to **module.exports**.
- The **module**.**exports** in the above example **exposes** a log object as a module.

- The **module.exports** is a special object which is included in every **JS file** in the Node.js application by default.

➢ Loading Local Module

- To use local modules in our application, we need to load it using **require()** function in the same way as core module.
- **File : app.js**

```
var myLogModule = require('./Log.js');

myLogModule.info('Node.js started');
```

- In the above example, **app.js** is using log module. First, it loads the logging module using **require()** function and specified path where logging module is stored.
- Logging module is contained in **Log.js** file in the root folder. So, we have specified the path **'./Log.js'** in the require() function. The '**.**' denotes a root folder.
- The require() function returns a log object because logging module exposes an object in Log.js using **module.exports**.
- Now we can use logging module as an object and call any of its function using dot notation e.g **myLogModule.info()** or **myLogModule.warning()** or **myLogModule.error().**
- Thus, we can create a local module using **module.exports** and use it in our application.

5.3 Built-in Modules

- Node.js has a set of built-in modules which we can use without any further installation.
- Here is a list of the built-in modules of Node.js

Module	Description
buffer	To handle binary data
cluster	To split a single Node process into multiple processes
crypto	To handle OpenSSL cryptographic functions
dns	To do DNS lookups and name resolution functions
domain	Deprecated. To handle unhandled errors
events	To handle events
fs	To handle the file system

http	To make Node.js act as an HTTP server
https	To make Node.js act as an HTTPS server.
net	To create servers and clients
os	Provides information about the operation system
path	To handle file paths
querystring	To handle URL query strings
readline	To handle readable streams one line at the time
stream	To handle streaming data
string_decoder	To decode buffer objects into strings
timers	To execute a function after a given number of milliseconds
tls	To implement TLS and SSL protocols
url	To parse URL strings
util	To access utility functions
v8	To access information about V8 (the JavaScript engine)
vm	To compile JavaScript code in a virtual machine
zlib	To compress or decompress files

5.3.1 Require() function:

- We always use the NodeJS require() function to load modules in our projects.
- Require are used to consume modules. It allows to include modules in our programs.
- **Example:**

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

- To include a module, use the **require**() function with the name of the module.
- E.g. Now the application has access to the HTTP module, and is able to create a server.

5.3.2 User Defined Modules

- we can create own modules, and easily include them in applications.
- **Example:** Create a module that returns the current date and time.

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

- Use the **exports** keyword to make **properties** and **methods** available outside the module file.
- Save the code above in a file called "**myfirstmodule.js".**

➢ Include Your Own Module

- We include and use the module in any of our Node.js file.
- **Example:** Use the module "**myfirstmodule**" in a Node.js file.

```
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);
```

5.4 Node.js as Web Server

- ✓ To access web pages of any web application, we need a web server. The web server will handle all the http requests for the web application.
- ✓ e.g. IIS is a web server for ASP.NET web applications and Apache is a web server for PHP or Java web applications.
- ✓ Node.js provides capabilities to create our own web server which will handle HTTP requests asynchronously.
- ✓ We can use **IIS** or **Apache** to run **Node.js web application** but it is recommended to use **Node.js web server**.

5.4.1 Create Node.js Web Server

- ✓ Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.
- ✓ **Example:** simple Node.js web server contained in **server.js** file.

```
var http = require('http');     // 1 - Import Node.js core module
var server = http.createServer(function (req, res) {   // 2 - creating server
        //handle incomming requests here..
});
server.listen(5000); //3 - listen for any incoming requests
console.log ('Node.js web server at port 5000 is running...')
```

- In the above example, we **import** the http module using **require**() function.
- The http module is a core module of Node.js.
- The next step is to call **createServer()** method of http and specify callback function with **request** and **response** parameter.
- Finally, call **listen**() method of server object which was returned from **createServer**() method with port number, to start listening to incoming requests on port 5000.
- we can specify any unused port here.
- Run the above web server by writing node server.js command in command prompt or terminal window.

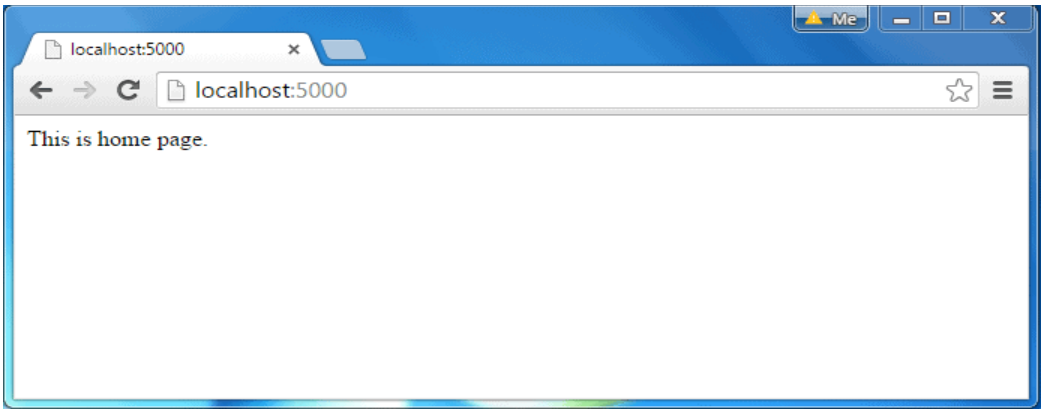
❖ Handle HTTP Request

- ✓ The **http.createServer()** method includes request and response parameters which is supplied by Node.js.
- ✓ The request object can be used to get information about the current HTTP request e.g., url, request header, and data.
- ✓ The response object can be used to send a response for a current HTTP request.
- ✓ **Example:** Demonstrates **handling HTTP request and response** in Node.js.

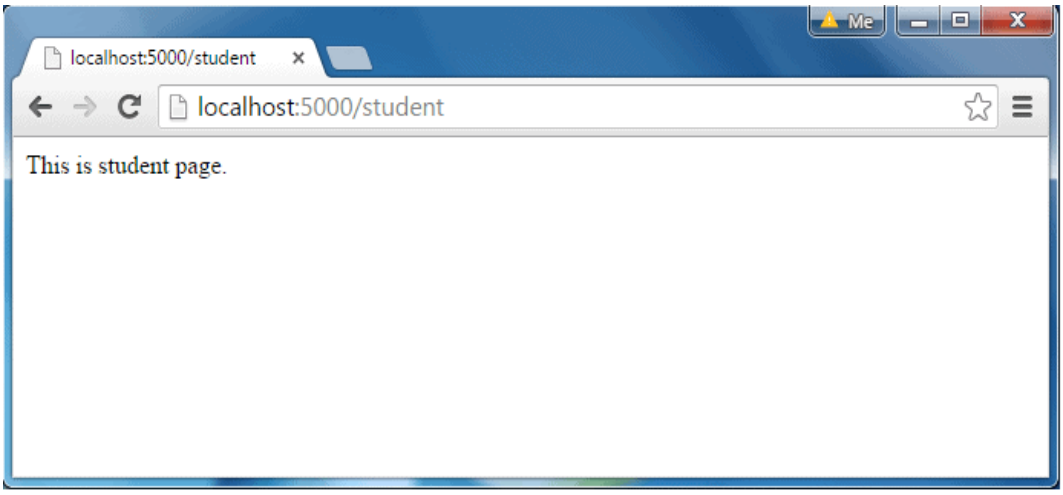
```
var http = require('http');   // Import Node.js core module
var server = http.createServer(function (req, res) {   //create web server
   if (req.url == '/') {  //check the URL of the current request
                          // set response header
      res.writeHead(200, { 'Content-Type': 'text/html' });  // set response content
      res.write('<html><body><p>This is home Page.</p></body></html>');
      res.end();
```

```
    }
    else if (req.url == "/student") {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student 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(5000);   //6 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

- In the above example, **req.url** is used to check the **url** of the current request and based on that it sends the response.
- To send a response, first it **sets the response header** using **writeHead**() method and then **writes a string as a response body using write()** method.
- Finally, Node.js web server **sends the response** using **end()** method.
- For Windows browse **http://localhost:5000** and see the following result.



- The same way, point our browser to **http://localhost:5000/student** and see the following result.



➢ Sending JSON Response

✓ The following example demonstrates how to serve JSON response from the Node.js web server.

✓ **Example:**

```
const http = require('http');
const responseData = {
        message:"Hello, Students ",
        articleData:{
                articleName: "How to send JSON response from NodeJS",
                category:"NodeJS",
                status: "published"
        },
        endingMessage:"Visit NodeJS Website for more Information"
}
const server = http.createServer(function(req,res){
        console.log("Request is Incoming");
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify(responseData));
```

```
});
server.listen(3000);
console.log("Server is Listening at Port 3000!");
```

5.4.2 What is Query String

- ✓ A query string is a part of the uniform resource locator (URL), that assigns values to specified parameters. In plain text it is the string after the ' **?** ' in a URL.
- ✓ It is meant to send small amounts of information to the server via the URL. This information is usually used as parameters to query a database, or maybe to filter results.
- ✓ Some URL with Query String examples are shown below.
  - https://example.com/over**?name=samir**
  - https://example.com/product**?id=123&color=purple**

❖ Node.js Query string module

- ✓ 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, we need to use **require('querystring')**.
- ✓ The Node.js query string module provides methods for parsing and formatting URL query strings.
- ✓ Another way to access query string parameters is parsing them using the **querystring builtin Node.js module**.

❖ Query String Methods

Method	Description
escape(str)	Returns an escaped querystring
parse()	Parses the querystring and returns an object
stringify()	Stringifies an object, and returns a query string
unescape()	Returns an unescaped query string

➢ Node.js Query String: parse()

- The **querystring.parse()** method is used to parse the URL query string into an object that contains the key value pair.
- The object which we get is not a JavaScript object, so we cannot use Object methods like obj.toString.

- **Example:**

```
const querystring = require('querystring');
const obj1=querystring.parse('name=sonoo&company=javatpoint');
console.log(obj1);
```

➢ Node.js Query String: stringify()

```
const querystring = require('querystring');
const qs1=querystring.stringify({name:'sonoo',company:'javatpoint'});
console.log(qs1);
```

➢ Access query string parameters

- In Node.js, functionality to aid in the accessing of URL query string parameters is built into the standard library.
- **Example:** app.js

```
var http = require('http');
const url=require('url');
var server = http.createServer(function(req, res) {
    var string=url.parse(req.url,true).query;
    //var folder=url.parse(req.url).pathname;
    res.write(string.id+" "+ string.id);
    res.end();
    //console.log(url.parse(req.url));
    //console.log(url.parse(req.url).pathname);  //  /user
    //console.log(url.parse(req.url).query);   //id=101&name=jatin
});
server.listen(3000);
```

- The **url.parse()** method returns an object which have many key value pairs one of which is the query object.

- The **url.parse()** method returns an object which have many key value pairs one of which is the query object.
- Some other handy information returned by the method include host, pathname, search keys.
- To test this code run node app.js (app.js is name of the file) on the terminal and then go to your browser and type **http://localhost:3000/user?id=101&name=jatin** on the URL bar

✓ url.parse(req.url,true).query

returns { id: '101', name: 'jatin' }.

✓ url.parse(req.url,true).host

returns 'localhost:3000'.

✓ url.parse(req.url,true).pathname

returns '/user'.

✓ url.parse(req.url,true).search

returns '?id=101&name=jatin'.

5.5 File System Module:

✓ The Node.js file system module allows to work with the file system on computer.
✓ To handle file operations like creating, reading, deleting, etc., Node.js provides an inbuilt module called **fs (File System).**
✓ All file system operations can have synchronous and asynchronous forms depending upon user requirements.
✓ **Syntax: var fs = require("fs")**

❖ Common use for the File System(FS) module:

1. **Read files**
2. **Create files**
3. **Update files**
4. **Delete files**
5. **Rename files**

➢ Important method of fs module

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.**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.**exists**(path, callback)	Determines whether the specified file exists or not.
fs.**appendFile**(file, data[, options], callback)	Appends new content to the existing file.

1. Reading File

- ✓ Use **fs.readFile()** method to read the physical file asynchronously.
- ✓ **Syntax:** fs.readFile(fileName [,options], callback)
- ✓ **Parameter Description:**
  - **filename**: Full path and name of the file as a string.
  - **options:** The options parameter can be an object or string which can include encoding and flag. The default encoding is **utf8** and default flag is **"r".**
  - **callback:** A function with two parameters **err** and **data**. This will get called when readFile operation completes.
- ✓ **Example :** The following example demonstrates reading existing **TestFile.txt** asynchronously.

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

- The above example reads **TestFile.txt** (on Windows) asynchronously and executes callback function when read operation completes.
- This **read operation either throws an error or completes successfully**. The **err** parameter contains error information if any.
- The **data** parameter contains the content of the specified file.

✓ Use **fs.readFileSync()** method to read file synchronously as shown below.

✓ **Example:** Reading File Synchronously

```
var fs = require('fs');
var data = fs.readFileSync('dummyfile.txt', 'utf8');
console.log(data);
```

2. Write File

✓ Use **fs.writeFile()** method to write data to a file.

✓ If file already exists, then it overwrites the existing content otherwise it creates a new file and writes data into it.

✓ **Syntax: fs.writeFile(filename, data[, options], callback)**

✓ **Parameter Description:**

- **filename:** Full path and name of the file as a string.
- **Data:** The content to be written in a file.
- **options:** The options parameter can be an **object** or **string** which can include encoding, mode and flag. The **default encoding is utf8 and default flag is "r".**
- **callback:** A function with two parameters **err** and **fd**. This will get called when write operation completes.

✓ **Example:** The following example creates a **new file called test.txt** and **writes "Hello World"** into it asynchronously.

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

3. Append File

- ✓ Use **fs.appendFile()** method to append the content to an existing file.
- ✓ **Example:** Append File Content

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

4. Open File

- ✓ Alternatively, we can open a file for reading or writing using **fs.open**() method.
- ✓ **Syntax: fs.open(path, flags[, mode], callback)**
- ✓ **Parameter Description:**
  - **path:** Full path with name of the file as a string.
  - **Flag:** The flag to perform operation
  - **Mode:** The mode for **read, write or readwrite**. **Defaults to 0666 readwrite.**
  - **callback:** A function with two parameters **err** and **fd**. This will get called when file open operation completes.
- ✓ **Flags :** Following flags which can be used in read/write operation.

Flag	Description
r	Open file for reading. An exception occurs if the file does not exist.
r+	Open file for reading and writing. An exception occurs if the file does not exist.
rs	Open file for reading in synchronous mode.
w	Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
w+	Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
a	Open file for appending. The file is created if it does not exist.
a+	Open file for reading and appending. The file is created if it does not exist.

- ✓ **Example**: The following code open a file input.txt for reading and writing.

```
var fs = require("fs");
fs.open('input.txt', 'r+', function(err, fd) {
    if (err) {
        return console.error(err);
     }
    console.log("File opened successfully!");
});
```

5. Delete File

- ✓ Use **fs.unlink()** method to delete an existing file.
- ✓ **Syntax : fs.unlink(path, callback);**
- ✓ **Parameter Description:**
  - **path:** Full path with name of the file as a string.
  - **Callback:** This is the callback function No arguments other than a possible exception are given to the completion callback.
- ✓ **Example:** The following example deletes an existing file.

```
var fs = require('fs');
fs.unlink('test.txt', function () {
    console.log('write operation complete.');
});
```