Assignment :- 1

Name :- ROHAN Jwala.

Roll No :- 18

Subject :- Full stack development

Semester :- 7th Sem

Date :- 21/07/2023

Q.1 Nodejs : Introduction, Features, execution architecture.

→ Node.js is a cross-platform runtime environment and library for running Javascript application 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 application.

→ Node.js also provides a rich library of various Javascript modules to simplify the development of web Applications.

* Nodejs Features

1] Extremely Fast :-
Node.js is built on google chrome's v8 Javascript Engine. so its library is very fast in code execution.

2] 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 Event 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.

3] Single threaded :-

Node.js follows a single threaded model with event looping.

4] Highly scalable :-

Node.js ~~cuts down~~ is highly scalable because event mechanism helps the server to respond in a non-blocking way.

5] No buffering :-

Node.js cuts down the overall processing time while uploading audio and video files. Node.js Application never buffers any data. The Application simply output the data in chunks.

6] Open Sources :-

Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js Applications.

7] License :-

Node.js is released under the MIT license.

* Node.js Execution Architecture

1] V8 Javascript Engine :

Node.js is built on top of the V8 Javascript engine, which provides the runtime environment for executing Javascript code. V8 compiles Javascript code into machine code, making it highly performant.

2] Event Loop :

Node.js utilizes an event loop to handle I/O operations asynchronously. The event loop continuously checks for pending events in the event queue and executes their associated callback functions.

3] Single Threaded :-

Node.js runs JavaScript code in a single-threaded event loop. This means that all JavaScript code execution is done on a single thread.

→ I/O operations, such as reading from file or making network requests, are handled asynchronously by the event loop.

4] Non-blocking I/O :-

Node.js uses non-blocking I/O operations to perform tasks like file system operations, network requests and database queries. Instead of waiting for these operations to complete, Node.js delegates them to the operating system and continues executing other tasks.

5] Modules :-

Node.js follows the commonJS module system, allowing developers to create reusable modules in separate files. Modules encapsulate code and provides a clean way to organize and structure applications.

Q.2 Note on modules of Node.js with examples.

→ In node.js application, a module can be considered as a block of code that provide a simple or complex functionality that can communicate with external Application.

→ module can be organized in a single file or a collection of multiple files / folders.

→ Almost all programmers prefer modules because of their reusability throughout the Application and ability to reduce the complexity of code into smaller pieces.

→ Node.js uses the common.JS module standard implementation in its module ecosystem.

→ There are three type of modules in Node.js

1] core modules
2] Local modules
3] third-party modules.

1) Core modules :-

Node.js comes with dozen of built in modules. These built in modules are sometimes refered to as core modules.

→ The module system is built in around the require function, this function is used to load module and get access to its controls.

→ Require is a global variable provided to all your Node.js scripts. So you can use it anywhere you like.

- Require (require() function will return a Javascript type depending on your module.

Syntax

```
Const module = require ('module-name');
```

→ You can directly use the node.js core module in your application without any initialization

Example

```
Const Fs = require ('fs'); // Import Fs module
Fs.writeFileSync ('abc.txt', 'I Love nodejs');
```

→ Core modules Name

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

os	It provides information about the operating system.
Path	To handle File Path
cluster	To split a single Node process into multiple process.
dns	To do DNS lookups and name resolution Functions.
tls	To implement TLS and SSL protocols.
querystring	To handle URL querystring
url	To parse URL string
events	To handle events
timers	To execute a function after a given number of ms.

2] Local module :-

Putting all your code in a single file is not a good idea.

→ As you add more code, you will want to stay organized and break your Node.js app into multiple scripts that all work together. For that purpose, we can create local modules in our applications.

Exporting From Files

First of all we need to create a file called myFunc.js file.

Now we can export javascript code written on this file using module.exports.

myFunc.js File

```
const Func = Function () { console.log ("Hello, module");
};
module.exports = Func;
```

Importing our Files :-

Moreover, we need to import the local file into index.js

index.js File

```
const myFunc = require ('./myFunc.js');
myFunc();
```

output

Hello, this is module

3/ Third - Party modules :-

third-party modules can be installed from the npm (Node package manager) available online.

→ To use it, Firstly we need to initialize the npm using npm init command and it will create Package.json File in the root directory.

→ It stores the all information about the third-Party modules that we have installed as a dependency.

Installing an Npm module

npm install "module-name"

Ex: npm install express

npm i validator.

Q.3 Note on Package with Example.

→ A Package in node.js contains all the files you need for a module.

→ modules are Javascript libraries you can include in your project.

→ Download a Package

Open the command line interface and tell Npm to download the Package you want.

I want to download a package called "upper-case"

npm install upper-case

→ Npm create a Folder named "node_modules", where the package will be placed. All Packages you install in the future will be placed in this Folder.

→ using a Package.

Include the "upper-case" package the same way you include any other module.

```
Var uc = require("upper-case");
```

Example

```
Var http = require("http");
Var uc = require("upper-case");
http.createServer((req,res)=>{
  res.writeHead(200, {"content-Type": "text/html"});
  res.write(uc.upperCase("Hello world"));
  res.end();
}).listen(8000);
```

Q.4 Use of Package.json and package-lock.json

→ Package.json :-

In Node.js the package.json files is a critical files that serves multiple purposes and is commonly used in Node.js Projects. It is essentially a metadata file for the project, and it is typically located in the root directory of the project.

→ The file wrote in JSON format and includes the following information:

→ Project name and description

→ Author and contributor

→ version of the projects

→ List of dependencies required to run the project

→ List of development dependencies used during development but not required for production

→ Scripts to run various tasks, start the application, running tests etc.

→ **Package-lock.json**:

Package-lock.json is an automatically genrated file that provides a detailed description of the exact versions of all dependencies installed in the project.

→ This file helps ensure that everyone working on the projects, regardless of their environment install the same versions of the dependencies. It also locks the dependency tree to avoid introducing discrepancies between different installations.

→ Workflow of Node.js Project

(1) Developers defines the project's dependencies in the 'package.json' file.

(2) Other developers or build system run 'npm install' to install dependencies listed in 'package.json'. This creates or update the 'node_modules' folder in the project.

(3) When 'npm install' runs, it reads 'package-lock.json' to determine the exact version of the dependencies to install, ensuring consistency across different installations.

→ 'package.json' is used to defining project metadata and listing dependencies, while 'package-lock.json' ensure consistent dependency installation accross different environment by locking versions.

Q.5 Node.js Packages

→ Node.js Packages are reusable modules of code that can be easily installed and used in node.js projects to add specific functionalities or features.

→ They are distributed through the node package manager (NPM) or yarn and be easily installed into a project using a package manager.

→ Node.js Packages can be categorized into two main types.

1) Core module :

There are built in modules that come in with Node.js and can be used without requiring any additional installation. Core modules are typically used for fundamental tasks like file system operations, HTTP server creation, and handling operating system-related functionality. For example, 'fs' for file system operations, 'HTTP' for creating HTTP servers, and 'os' for interacting with the program operating system.

2] Third Party modules

→ These are external packages created by the Node.js community and other developers, which provide additional functionality beyond the core module. To use third-party module, you need to install them using a package manager like npm or yarn. The package manager reads the dependencies listed in the 'package.json' file and installs them in the 'node_modules' directory.

→ To install Node.js package using npm, you would run the following command in the terminal.

npm install package-name.

→ Third party Node.js packages cover a wide range of functionalities, including web framework, database controls, utility libraries, authentication modules, testing frameworks, and much more.

- Express: A popular web application framework for building webserver and APIs.

→ lodash :- A utility library with various helper functions for working with arrays, object and more.

→ mongoose :- An ORM (object - Relational mapping) library for mongo.DB

→ Axies :- A popular HTTP client for making API requests.

Q.6 Npm introduction and commands with its use.

→ Initialize a new projects :-

To create a new node.js project, you can initialize it with a 'package.json' file using following command:

* npm init

→ This command will guid you through a series of prompts to set up the project details like name, version, description, entry point etc.

Installing Packages :

To install Packages and add them as dependencies to your project, you can use the 'npm install' command.

* npm install package-name

This will install the specific package and save it as a runtime dependency in your 'package.json'.

SAVE the Package :-

If you want to save Package only for development purpose you can use '--save-dev'.

* npm install package-name --save-dev

Install Package globally :-

Some packages are designed to be used globally across projects. You can install them globally using the '-g' flag.

npm install -g package-name

it may cause version conflicts between projects.

→ Uninstall Packages :

To Remove a package from your Project's dependencies, 'npm uninstall' command :

npm uninstall Package-name

→ Update Packages :-

To update packages to their latest complete version, you can use the 'npm update' command :

npm update

→ This will update all package in 'node-modules' directory to their latest-compatible versions, based on the version ranges specified in your 'package-json' file.

→ Search For Packages :

To search for package available on the Npm registry, you can use :

npm search package-name.

→ View installed Packages :-

To view list all packages installed in your projects.

npm list

For more concise view use: npm list --depth=0

Q.7 Describe the Node.js Packages

1) URL :

In Node.js a URL is a references to a resource on the web or on a local file system.

→ It is used to identify the location of a specific resources such as web page or a file.

→ The main module for working with URLs in Node.js is the 'url' module.

→ Use of URL module

(1) Parsing URL :-

You can parse a URL string into its components using the 'url.parse()' method.

(2) Formatting a URL :-

If you have the components of URL and want to create a URL string, you can use the 'url.format()' method.

* Properties

1. 'url.protocol' :: The protocol scheme of the URL (eg) = http, https

2. 'url host' :: The full host portion of the URL, including the hostname and port.

3. 'url hostname' :: The hostname portion of the URL

4. 'url port' :: The port number specified in the URL.

5. 'url.pathname' :- The path portion of the url.

Ex
```
const url = require ('url');
const urlstring = 'https://www.example.com:8000/
path/to/resource?query=param';
const parsedurl = url.parse (urlstring, true);
console.log (parsedurl.protocol);
console.log (      "      .host);
console.log (      "      .hostname);
console.log (parsedurl.query);
```

2| Node.js Process

→ Node.js provides the facility to get process information such as process id, architecture, platform, version, release, uptime, CPU usages etc.

→ The process is global object an instance of EventEmitter → can be accessed from anywhere.

* Properties

(1) arch :- Returns process architecture : 'arm'

(2) args :- returns commands line arguments as an array.

(3) env :- returns user environment

(4) pid :- returns process id of the process

* Node.js process methods

(1) cwd() :- Returns path of current working directory.

(2) hrtime() :- Returns the current high resolution real time in a array.

(3) memoryUsage() :- returns an object having information of memory usage.

(4) uptime() :- Returns the Node.js process uptime in seconds.

Ex
```
process.on('exit', function(code) {
    setTimeout(function() {
        console.log("This will not run");
    }, 0);
    console.log('About to exit with code:', code);
});
console.log("Program ended");
```

3] Node.js Readline

→ The Readline module provides a way of reading a data stream, one line at a time.

→ The syntax

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

→ Readline properties and methods

(1) clearLine() :- clears the current line of the specified stream.

(2) clearScreenDown() :- clears the specified stream from the current cursor down position.

(3) createInterface() - creates an Interface object.

(4) cursorTo() :- Moves the cursor to the specified position.

Ex

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

var myInterface = readline.createInterface({
    input :
fs.createReadStream('demoFiles.html')});

var lineno = 0;
myInterface.on('line', function(line)
{
    lineno++;
    (console.log('line number ' + lineno + ":" +
line));
});
```

4] Node.js Event

→ Every action on a computer is an event. like when a connection is made or a file is opened.

→ object in Node.js can fire events, like the readstream object fires events when opening and closing a file:

→ Events module

→ 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.

Example

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

5] Node.js Console

→ The Node.js console module provides a simple debugging console similar to JavaScript console mechanism provided by web browsers.

→ Three method of console.

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

→ (1) console.log() is used to display simple message on console.

```
console.log('Hello JavaTpoint');
```

→ (2) Node.js console.error() :- The console.error() Function is used to render error message on console.

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

→ (3) Node.js console.warn() :- The console.warn() Function is used to display warning message on console.

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

6) Node.js Buffer

→ The buffer module provides a way of handling streams of binary data.

→ The Buffer object is a global object in Node.js and it is not necessary to important it using the require keyword.

Syntax: var buf = Buffer.alloc(15);

→ Buffer Properties and methods

(1) alloc() :- creates a Buffer object of the specific length.

(2) allocUnsafe() :- creates a non-zero-Filed Buffer of the specified length.

(3) compare() :- compares two Buffer objects.

(4) byteLength() :- Returns the numbers of bytes in a specified object.

example var buf = Buffer.from('abc');
console.log(buf);

7) querystring

→ The Query string module provides a way of Parsing the URL query string.

Syntax

var querystring = require('querystring');

→ method

(1) escape() :- Returns an escaped querystring.

(2) Parse() :- Parse the querystring and returns an object.

(3) stringify() :- Stringifies an object, and returns a querystring.

(4) unescape() :- Returns an unescaped query string.

ex
```
var qstring = require ('querystring');
var q = querystring.parse ('year :2017 & month = feb');
console.log (q.year);
```

8] HTTP in Node.js

→ To make HTTP Requests in Node.js there is a built-in module HTTP in Node.js to transfer data over the HTTP. To use the HTTP server in the node, we need to require the HTTP module. The HTTP module creates an HTTP server that listens to server that listens to server ports and gives a response back to the client.

Syntax

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

ex
```
const http = require ('http')
http.createServer ((request, response) => {
    response.write ('Hello World!');
    response.end();
}).listen (3000);
```

9] Node.js V8

→ The Node.js V8 module represents interface and event specific to the version of V8. It provides method to get information about heap memory through v8.getHeapStatics() and v8.getHeapSpaceStatistics() method.

Syntax: const v8 = require('v8');

→ v8.getHeapStatistics() :- The v8.getHeapStatistics() method returns statistics about heap such as total heap size, used heap size, heap size limit, total available size etc.

```
const v8 = require('v8');
console.log(v8.getHeapStatistics());
```

10] Node.js OS module

→ The OS module provides information about the computer's operating system.

→ Sy `var os = require('os');`

* Properties and method

(1) arch() :- Returns the os cpu architecture.

(2) cpus() :- Returns an array containing info about the computer's cpus

(3) endianness() :- Returns the endianness of the cpu.

(4) constant :- Returns an object containing the operating system's constant for process signals, error codes etc.

(5) freemem() :- Returns the number of free memory of the system.

Example

```
var os = require('os');
console.log("platform:" + os.platform());
console.log("Architecture:" + os.arch());
```

12) zlib :-

The zlib module provides a way of zip and unzip files.

Syntax var zlib = require ('zlib');

* Method and Properties

(1) Constants :- Returns an object containing zlib constant.
(2) createDeflate() :- Creates a Deflate object
(3) createGunzip() :- Creates a Gunzip object.
(4) createGzip() :- Creates a Gzip object.
(5) deflate() :- Compress a string or buffer, using Deflate.
(6) gzip() :- Compress a string or buffer, using Gzip.

Eg var zlib = require ('zlib');
var fs = require ('fs');
var gzip = zlib.createGzip();
var r = fs.createReadStream ('./demofile.txt');
var w = fs.createWriteStream ('./myzipfile.txt.gz');
r.pipe(gzip).pipe(w);