Name: Gheewala Poojan Dilipbhai

Roll No: 017

Subject : Application Development using Full Stack [701]

Semester : 7th

Theory Assignment - 1

①

Nodejs : Introduction, features, execution and architecture

=> Introduction to Node.js:

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to build server-side and networking applications. It is built on the V8 Javascript engine, the same engine that powers Google Chrome, which enables Node.js to execute Javascript code outside of a browser environment. This makes Node.js an excellent choice for developing scalable and high-performance applications that can handle a large number of concurrent connections.

=> Features of Node.js:

① Asynchronous and Event Driven: All APIs of Node.js library are asynchronous that is non-blocking. It essentially means 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.

② Very Fast: Being built on Google Chrome's V8 Javascript Engine, Node.js library is very fast in code execution.

(2)

(3) Single threaded but highly scalable:
Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP server.

(4) No Buffering: Node.js applications never buffer any data. These applications simply output the data in chuncks.

(5) License: Node.js is released under the MIT license.

⇒ Node.js Execution Architecture: Based on the event-daiven, non-blocking I/O model. Here's an overview of how it works.

(1) Event Loop: The event loop is that at the core of Node.js execution architecture. It constantly checks the event queue of pending events and executes the corresponding callback functions.

(2) Event Queue: Asynchronous operations, such as file I/O, network requests, or timers are initiated by Node.js. when these operations are completed, their corresponding callback functions are placed in the event queue.

(3) callback functions: Callback functions are associated with asynchronous operations. when an asynchronous operation completes, the corresponding

③ callback functions is added to the event queue.

④ Call stack : The call stack is a data structure that tracks the execution of functions in the program. When the event loop executes a callback function from the event queue, it is added to the call stack for execution.

⑤ Non-Blocking I/O : Node.js uses non-blocking I/O operations to perform tasks efficiently.

Node.js 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 implements CommonJs modules standard. CommonJs is a group of volunteers who define Javascript standards for web server, desktop, and console application.

- Node.js Module Types:

① Core Modules
② Local Modules
③ Third party Modules

(4)

(1) Node.js Core Modules:

Node.js is a light weight framework. The core modules include base minimum functionalities of Node.js. These core modules are compiled into it's binary distribution and load automatically when Node.js process starts. However, you need to import the core modules first in order to use it in your application.

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 programmers.

Eg

```
var module = require('module_name');
```

(2) Node.js Local Module:

Local modules are modules created locally in your node.js application. These modules includes different functionalities of your applications in separate files and folders. You can also package

VISION

it and distribute it via NPM, so that Node.js community can be use it. For example, if you need to connect to mongo db. and fetch data then you can create a module for it, which can be reused in your application.

Example

Let's write simple logging module which logs the information, warning or error to the console.

```
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
```

- Loading Local Modules:

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

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

(6)

=> Export module in Node.js:

The module.exports is a special object which is included in every Javascript file in the Node.js application by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module. So, whatever you assign to module.exports will be exposed as a module.

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');
```

(7)

Example

```
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 (8000);
```

Use of package.json and package-lock.json:

=> 'package.json' and 'package-lock.json' are two important files used in Node.js projects for managing dependencies and ensuring consistency between different installations of the project. Let's take a closer look at each file and their respective purposes:

- 'Package.json': is a metadata file for Node.js projects that contains information about the project and its dependencies. It is typically located at the root of project directory. The file wrote in JSON format and includes the following information:
  - Project name and description
  - Author and contributors
  - version of the project
  - 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.

(8)

- 'package-lock.json':
  Introduced in npm version 5, 'package-lock.json' is an automatically generated 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 project, regardless of their environment, installs the same versions of the dependencies. It also locks the dependency tree to avoid introducing discrepancies between different installations.

=> The 'package-lock.json' file should be commited to version control to enable consistent installations across different environment.

- workflow of Node.js project:

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

(2) Other developers or build system run 'npm install' to install the dependencies listed in 'package.json'. This creates or updates the 'node_modules' folder in the project directory, containing all the installed packages.

(3) When 'npm install' runs, it reads 'package-lock.json' to determine the exact versions 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' ensures consistent dependency installation across different environments by locking versions.

VISION

(9)

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 can be easily installed into a project using a package manager.

- Node.js packages can be categorized into two main types.

(1) Core modules: These 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 functionalities. 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 modules, 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.

VISION

- To install a 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 frameworks, database connectors, utility libraries, authentication modules, testing frameworks and much more.
  - Express: A popular web application framework for building web servers and APIs.
  - lodash: A utility library with various helper functions for working with arrays, objects and more.
  - Mongoose: An ORM (Object-Relational Mapping) library for Mongo.DB.
  - Axios: A popular HTTP client for making API requests.
  - Jest: A widely used testing framework for Node.js applications.

NPM introduction and commands with its use.

=> Initialize a New Project:

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

```
npm init
```

command will guide 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 specified package and save it as a runtime dependency in your 'package.json'.

=> Save the Package:

If you want to save the package only for development purposes you can use '--save-dev':

npm install package-name --save-dev

=> Install packages 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.

(12)

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

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

=> Install packages from 'package.json':
To install all dependencies listed in your 'package.json', you can use:

npm install
This command reads the 'package.json' file and installs all the dependencies listed their 'dependencies' and 'devDependencies'.

=> Search for Packages:
To search for packages available on the NPM registry, you can use:

npm search package-name

=> View installed Packages:
To view list all packages installed in your project,

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

vision

Node.js URL module:

- The URL module splits up a web address into readable parts.
- To Include the URL module, use the require() method.

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

- Parse an address with me the url.parse() method, and it will return a URL object with each part of the address as properties:

Example

```
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);
console.log (q.pathname);
console.log (q.search);

var qdata = q.query;
console.log (qdata.month);
```

Node.js process, pm2 :

- pm2 (Process Manager e) is an external package used to manage and monitor Node.js applications. It helps you easily manage your Node.js processes by providing features like automatic restarts, clustering, log management, and more. pm2 is

particularly useful for deploying and maintaining Node.js applications in production environments.

① Install PM2:

First, you need to install PM2 globally on your system. Open a terminal or command prompt and run the following command.

```
npm install -g pm2
```

② start an application with pm2:

To start a Node.js application with PM2, navigate to the directory where your Node.js application is located. Then use 'pm2 start' command to start the application:

```
pm2 start app.js
```

③ Manage Applications:

You can use various PM2 commands to manage your applications. Here are some of the commonly used ones:

'pm2 list' - list all running applications manage by pm2

'pm2 stop <app_name|app-id>' - stop a applicatio

'pm2 restart <app_name|app-id>' - Restart a specific Application

'pm2 delete <app_name|app-id>' - delete a specific application from pm2's management.

④ Save and Restore Process list:

PM2 can save the list of managed processes to a file.

```
pm2 save
```

To restore the saved process

```
pm2 resurrect
```

⑤ Monitoring and Logs:

PM2 provides a monitoring Dashboard to check status of Application.

```
pm2 monit
```

PM2 manages logs of your Application.

```
pm2 logs <app-name | app-id>
```

⑥ clustering:

PM2 can also help in utilizing multiple CPU cores by running multiple instances of your application.

```
pm2 start app.js -i 4
```

This will run 4 instances of your Application.

Node.js Readline module:

The Readline module provides a way of reading a datastream, one line at a time.

- Syntax:

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

- Properties and methods

Method	Description
clearline()	clears the current line of the specified stream

clearScreenDown() => clears the specified stream from the current cursor down position.

createInterface() => Creates an Interface object

cursorTo() => moves the cursor to the specified position.

emitKeyPressEvents() => Fires keypress events for the specified steam

moveCursor() => moves the cursor to a new position, relative to the current position.

Example

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

var myInterface = readline.createInterface({
    input : fs.createReadStream ('demo file1.html');
});

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

Node.js fs :

The Node.js file system module allows you to work with file system on your computer.

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

- Common use for the File System module:
  - Read files
  - Create files
  - Update files
  - Delete files
  - Rename files

- Read files :

fs.readFile() method is used to read files on your computer.

Example

=> demofile1.html

```
<HTML>
<body>
<h1> Hey </h1>
</body>
</HTML>
```

=> Node.js file

```
var http = require ('http');
var fs = require ('fs');
http.createServer (function (req, res) {
    fs.readFile ('demofile1.html', function (err, data) {
    res.writeHead (200, {'content-HTML': 'text/HTML'});
    res.write (data);
    return res.end();
});}).listen (8080);
```

- create files:
  - fs.appendFile()
  - fs.open()
  - fs.writeFile()

Example

- appendFile()

```
var fs = require('fs');
fs.appendFile('mynewfile1.text', 'Hello Content',
        function (err) {
    if (err) throw err;
    console.log('saved!');
});
```

- fs.open()

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

- fs.writeFile()

```
var fs = require('fs');
fs.writeFile('mynewfile3.txt', 'HelloContent!', functi
                                        (err) {
    if (err) throw err;
    console.log('saved!');
});
```

- Delete Files:

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

Example

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

- Rename files:

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

Example

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

Node.js Events:

- Node.js is perfect for event-driven applications. Every action on a computer is an event. Like when a connection is made or a file is opened.
- objects in node.js can fire events, like the readstream object fires events when opening and closing a file.

- Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.

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

=> EventEmitter object:

- You can assign event handlers to your own events with the EventEmitter object.

Example

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

var myEventHandler = function () {
    console.log('I hear a scream!');
}
eventEmitter.on('scream', myEventHandler);

eventEmitter.emit('scream');
```

Node.js Console:

- Node.js Console is a global object and is used to print different levels of messages to stdout and stderr. There are built-in methods to be used for printing information, warning, and your error messages.

① console.log([data][,...])
Prints to stdout with newline. This function can take multiple arguements in a printf()-like way.

② console.info([data][,...])
Prints to stdout with newline. This function can take multiple arguements in printf() - like way.

③ console.error([data][,...])
Prints to stderr with newline. This function can take multiple arguements in a printf() - like way.

④ console.warn([data][,...])

⑤ console.dir(obj[,options])
Uses util.inspect on obj and prints resulting string to stdout.

⑥ console.time(label)
Mark a time

⑦ console.timeEnd(label)
Finish timer, record output

⑧ console.trace(message[,...])
Print to stderr 'Trace:', followed by the formatted message and stack here trace to the current position.

⑨ console.assert(value[,message])
Similar to assert.ok(), but the error message is formatted as util.format(message...);

Node.js Buffer Module:

- The buffers module provides a way of handling streams of binary data.
- The buffer object is a global object in Node.js and it it not necessary to import it using the require keyword.

```
var buf = Buffer.alloc(15);
```

Method	Description
Buffer.alloc(size)	=> It creates a buffer and allocates size to it.
Buffer.from(initialization)	=> It initializes the buffer with given data.
Buffer.write(data)	=> It writes the data on buffer.
toString()	=> It read data from buffer and returned it.
Buffer.isBuffer(object)	=> It checks whether the object is a buffer or not.
Buffer.length	=> It returns the length of the buffer.
Buffer.copy(buffer, subsection size)	=> copies data one buffer to another.
Buffer.slice(start, end = Buffer.length)	=> It returns the subsection of data stored in a buffer.
Buffer.concat([buffer.buffer])	=> It concatenates two buffers.

Node.js Query string :

The module provides a way of parsing the URL query string.

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

Method	Description
escape() =>	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

Example

```
var querystring = require ('querystring');
var q = querystring. parse ('year = 2017 & month
                             = february');
console.log (q.year);
```

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).

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

- HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- use the createServer() method to create an HTTP server.

- the HTTP server is supported to be displayed as HTML, you should include an HTTP header with the correct content-type:

Example

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

V8 engine:

=> V8 is the Javascript engine developed by Google, which is used in Node.js to execute Javascript code. Node.js uses V8 to run Javascript code on the server-side, allowing developers to build scalable and efficient network applications.

- Since V8 is continuously updated and improved, newer versions of Node.js often incorporates the latest V8 releases to take advantage of performance enhancements, bug fixes, and new language features, It is essential to keep your Node.js on up to date to benefit from the latest improvements.
- To find the latest version of Node.js and its corresponding V8 engine, you should visit the official Node.js website.

Node.js os module:

=> The os module provides information about the computer's operating system.

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

Method		Description
arch()	=>	Returns the operating system CPU architecture
constants	=>	Returns an object containing the operating system's constants for process signals, errors etc.
cpus()	=>	Returns an array containing information about the computer's CPUs
endianness()	=>	endianess of the CPU
EOL()	=>	end-of-line marker for the current operating system
freemem()	=>	number of free memory of the system
hostname()	=>	hostname of the operating system
loadavg()	=>	array containing the load averages
networkInterfaces()	=>	network Interfaces that has a network address
platform()	=>	information about the operating system's platform

release() => information about the operating System's please
tmpdir() => os default directory for temporary files
totalmem() => number of total memory of the system
type() => name of the operating system
uptime() => uptime of the os, in seconds
userInfo() => Information about the current user.

Example

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

Node.js zlib module:

zlib module provides a way of zip and unzip files.

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

Method	Description.
constants =>	object containing zlib constants
createDeflate() =>	Creates a Deflate object
createDeflateRaw() =>	Creates DeflateRaw object
createGunzip() =>	Creates Gunzip object
createGzip() =>	creates Gzip object
createInflate() =>	creates Inflate object

createInflateRaw() => create InflateRaw object

createUnzip() => Creates Unzip object

deflate() => Compress a string or buffer, using Deflate.

deflateSync() => Compress a string or buffer, synchronously, using Deflate.

deflateRaw() => Compress string or buffer, using deflateRaw.

deflateRawSync => compress a string or buffer, synchronously, using deflateRaw.

gunzip() => Compress a string or buffer, using Gunzip

gunzipSync() => Compress a string or buffer, synchronously, using Gunzip

Inflate() => Decompress a string or buffer, using Inflate

InflateSync() => Decompress a string or buffer, synchronously, using Inflate.

InflateRaw() => Decompress a string or buffer, using InflateRaw.

InflateRawSync() => Decompress a string or buffer, synchronously, using InflateRaw.

unzip() => Decompress a string or buffer, using Unzip.

unzipSync() => Decompress a string or buffer, Synchronously, using unzip.