Subject: Open Source web development.

Name: Rama Keyat.

Date 24/7/23
Page 1

Assignment=1 Roll No: 38.

* Node.js

→ Node.js is server side platform built on google chrom's Javascript engine (V8 engine).

→ It develop by Ryan Dahl in 2009.

→ Latest version is V0.10.36.

⇒ Node.js is an open source, cross platform runtime environment for developing server side and networking applications.

→ Node.js applications are written in Javascript and can be run within the node.js runtime on OSX, microsoft window and linux.

→ Node.js also provide a rich library of Java script module which simplifies the development of web Application using Node.js to a

great extent.

Node.js = Runtime Environment + JS library.

features

1) Asynchronous and Event Driven:
All API of Nodejs library are Asynchronous, that is non blocking it essentially means a nodejs based server never wait for an API to return Data. server move to next API after calling it and a Notification mechanism of Event of Nodejs helps the server to get a Response from the previous API call.

2) Very fast:
Built in google chrome V8 Javascript Engine, Node.Js library very fast in code execution.

3) Single Threaded but highly scalable.

Node.JS uses single threaded model with event looping Event mechanism helps the server to Response in a non-blocking way and making the server highly scalable as opposed to Traditional server 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 request than traditional server like Apache HTTP Server.

4) No Buffering:

Node.js application Never buffer any data. these application simply output the data in chunks.

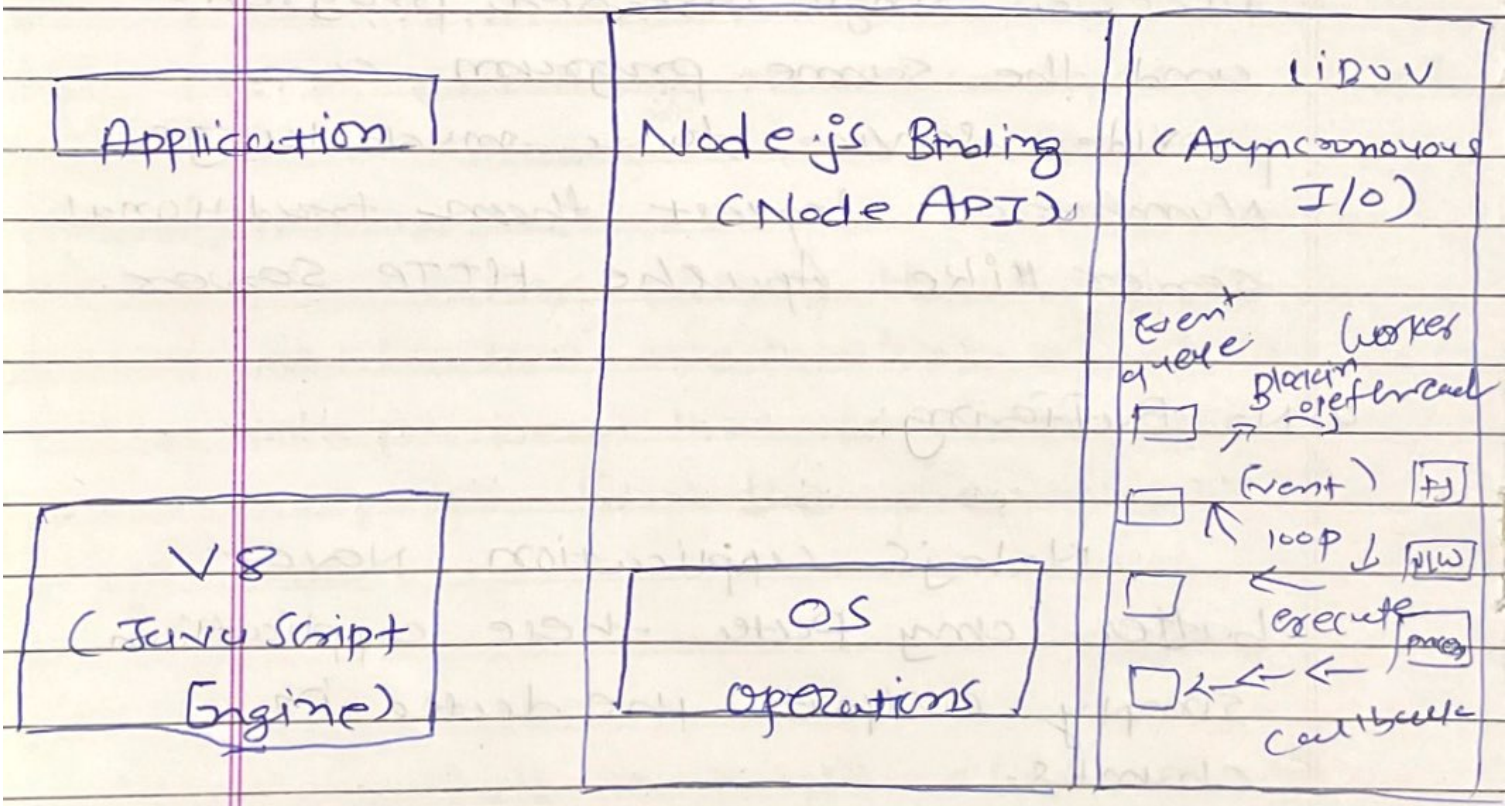
5) License:

Node.js is released Under

Under the MIT license.

2) Node.js execution architecture :-

⇒ Node.js follows a single-threaded, event driven and non-blocking I/O model, which allow it to handle a large Number of concurrent connections efficiently.



→ 1. V8 Engine:

Node.js relies on the V8 engine to interpret and execute Javascript code. V8 is written in c++ and is used by google chrome as well it compile Javascript code into Machine code using Just-in-Time Compilation, optimizing the code for performance.

2) Single threaded Event loop:-

This means that it uses a single thread to handle all incoming request and events. The event loop is at the Core of Node.js and enables asynchronous non blocking I/O operations.

3) Event-driven Architecture.

Node.js operates on an event-driven architecture. It allows developers to define callback functions that get executed when certain event occure, such as a

when file is read, a Network request Completes or a timer expire. the event driven approach is essential for non-blocking I/O operations.

4) Non-Blocking I/O :-

Node.js uses non-blocking I/O operations. which means when request is made to read or write data from a file or a Network, Node.js Doesn't wait for the operations to complete before moving on to the next task. instead, it continues executing other task and notified the appropriate callback when the I/O operation is finished.

5) Libuv :-

It is a multi platform C library that Node.JS Use to abstract the Underlying operating System's

I/O capabilities and provides a consistent API for asynchronous I/O operations. It is responsible for managing the Event loop, handling file system operations, timers, Network sockets and more.

6) Event queue.

When an asynchronous is completed, its corresponding callback is placed in the Event queue. The event loop continuously checks the event queue and execute the callbacks in First-in-first out (FIFO) manner.

7) Modules:

Node.js follows the CommonJS module system, allowing developers to organize their code into reusable modules. Each module encapsulates its own functionality and can be loaded into other modules using

the 'require' function.

* Concurrency model:-

Although Node.js runs on single thread, it can handle high level of concurrency due to its Non-Blocking Nature. instead of creating New thread of each incoming connection, Node.js's efficiently manage multiple connections Using a single threads making it highly scalable for applications that require handling a large Number of concurrently request.

* Note on module with example.

→ In node.JS modules are the blocks of encapsulated code that communicate with an external application on the basis of their related functionality.

→ Module can be single file or a collection of multiple files/folders.

→ The reason programmer are heavily Relevant on module is because of their reusability as well as the ability to break down a complex piece of code into manageable chunks.

→ Three types of Modules:

1) Core Modules
2) Local modules
3) Third-party modules.

1) Core Modules:

Node.js has many built in modules that are part of the platform and come with Node.js installation. These module can be loaded into a program by using the require function.

⇒Syntax:

```
const module ⇒ require ('Module-name');
```

⇒ The requires function will return a Javascript type depending on what particular module return the following ex demonstrate how to use the Node.JS http module to create a web server.

⇒example.

```
const http = require ('Http');
http- createserver (function (req, res){
    res. writehead (200, {content-type}' 8'
        text /html '});
```

res.write('Welcome to this page!');
res.end();
});
listen(3000);

Core modules:

http
assert - set of assertion function useful for testing
fs
Path
Process
os
querystring
url.

2) local modules:
Unlike built in and external modules local module are created locally in your Node.js application. Let's create a simple calculating module that calculates various operations.

→ Create a calc.js file that has

following code:

→ calc.js

```
exports.add = function (x, y) {
        return x + y;
};

exports.sub = function (x, y) {
        return x - y;
};

exports.mult = function (x, y) {
        return x * y;
};

exports.div = function (x, y) {
        return x / y;
};
```

→ index.js

```
const calculator = require('./calc');
let x = 50, y = 10;
```

```
console.log("Addition of 50 and 10 is"
        + calculator.add(x,y));

console.log("Abstraction of 50 and 10 is"
        + calculator.sub(x,y));

console.log("multiplication of 50 and 10
        is" + calculator.Mult(x,y));

console.log("division of 50 and 10 is"
        + calculator.div(x,y));
```

⇒ Third-party Modules:

Third-party modules are modules that are available using the Node package Manager (NPM). These modules can be installed in project folder or globally

→ Some of the popular third-party modules are mongoose, express, angular and React.

→ example.

- NPM install express
- npm install mongoose
- npm install -g angular/cli

* Note on package with example

→ NPM (Node package Manager)

→ It is a default package manager for Node.js and is written entirely in JavaScript.

→ develop by Base 2. schevete in Jan 19, 2010.

→ NPM manage all the packages and Modules for Node.js and consist of command line client NPM. It gets installed into the system with installation of Node.js.

→ Required Module and package

install by NPM.

→ A package contain all the files needed for a module and modules are the JS libraries that can be included in Node project according to the requirment all the dependencies of a project through the package.JSON files. It can also Update and Uninstall packages.

→ Some facts about NPM.

→ At time or writing article npm has 580096 registered package the average rate of growth of this number is 591/day while it outruns every other package Registry.

→ NPM is open source.

→ The Top npm package in the decreasing order are: lodash.

async, react, request, express.

-ex

npm i upper-case

```
var UC = require('upper-case');
var http = require('http');
http.createServer(function (req, res){
  res.writeHead(200, {'content-type':
        'text/HTML'})
  res.write(UC.upperCase("Hello"));
    res.end();
  }).listen(8080);
```

-output.

Hello.

Use of package.json and package lock.json.

→ Both package.json and package-lock.json are essential files used in Node.js project for managing dependencies. They are serve different purpose, and understanding their roles is crucial for proper dependency management.

1. package.json

→ The package.json files is the heart of a Node.js project. it is a JSON file that contain metadata about the project and it's dependencies. It's typically located in root Directory of the project. Some of the key information stored in package.json are,

→ project Information: Name Version desc, author license, etc.

→ Dependency: A list of external package required by the project To run correctly. These dependencies are specified with their package Name and Version.

→ DevDependencies: Similar to Dependencies but these package are only required during Development not for the production runtime.

→ Script: Custome script that can be execute using npm commands.

→ Configuration: any specific configuration related to the project or its dependencies.

2) package-lock.json.

→ The package.lock.json file is generated automatically by npm when you run the 'npm install' command.

→ It is used to lock-down the version of the installed package and their dependencies. The file ensures that the exact same version of packages are installed consistently across different environment.

→ The main purpose of package lock-json are:

→ 'Dependency Version locking':
It grants that the specific version of package used during Development are identical all team members and in production. This eliminates potential issue cause by different package versions.

→ Faster and Reliable installation
When 'npm install' is executed, npm reads 'package-lock-json' to determine the exact versions of package to install, which speed up the installation process and ensure reliability.

→ Security:
The lockfile includes checksums for each installed package, which helps to verify package integrity and enhance security by mitigating the risk of unauthorized package modification.

→ Use and working of following Node.js packages.

→ URL
The URL module provides utilities for parsing and formatting URL. it allow you to work with URLs, query string and URL components.

→ important properties and methods:
- URL.parse: parse an string and return object.

- URL.format: formats an object into a URL string.

→ URL.resolve: Resolve Target URL relative to base URL.

→ Parsing url and formatting URLs:

```
const URL = require("url");
const urlString = http://www.example.com:8080/path?query=string";

const parsedUrl = url.parse(urlString);
console.log(parsedUrl);
const formatted = url.format(parsedUrl);
console.log(formatted(URL);
```

2) process :-
use: The process module provides

information and control over the current Node.js process. it allows you to interact with the process environment, argument, standard, streams more

- imp property and method

- process.argv:- An array containing the command-line argument passed to Node.js process.

- process.env: An object containing the user environment.

- process.exit: Terminates the Node.js process with an optional exit code.

- process: Adds a listner function for a specific event.

3) pm2

- use: pm2 is an external package used to manage and monitor

Nodejs application in production. It provides feature like process management, clustering, Zero-downtime reload and log handling

→ Imp feature:

→ process management: starting, stopping, managing Node.js processes.

→ clustering: scaling application across multiple cores for improved performance.

→ Zero-downtime reload = allow updating application without any downtime

→ Log handling: (capture, manage and route). (pm2 log).

→ program:

- npm install -g pm2
- pm2 start app.js -

4) readline

Use: The 'readline' module provides an interface for reading data from a readable stream line-by-line.

→ Important properties & method :-

→ readline.createInterface: Create a new readline interface.

→ rl.one = adds a listner function for a specific event on the readline interface.

→ program.

```
Const readline = require('readline');
Const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
```

rl.question ('what is your name?:

(Name) => { console.log("Hello, {name}!");

rl.close();

});

5) fs.

Use: The 'fs' module provides a file system-related functionality allowing you to read, write and manipulated files and directions

→ important properties and methods:

→ fs.readfile: Asynchronously reads the contents of a file.

→ fs.writefile: Asynchronously write data to a file.

→ fs.readdir: Asynchronously reads the constents of a directory.

→ program.

```
const fs = require('fs');
fs.readFile('file.txt', 'utf-8',
  (err, data) => {
    if (err) throw err;
    console.log(data);
});

fs.writeFile('Newfile.txt', 'Hello,
  Node.js!', 'utf8', (err) => {
    if (err) throw err;
console.log('File written
    successfully');
});
```

6) Events.

Use: The 'events' module provides an event-driven architecture to handle and emit event within Node.js applications.

→ imp properties & methods:

→ events. EventEmitter: The class that enable event handling and emmiting.

→ emmiter.on : Adds a listner function for a specific

→ emmitter.emit : Emits an event with optional Date arguments.

→ program:

```
const (EventEmmitter = require('events');

class myEmitter extends eventEmmiter {
const myEmitter = new MyEmitter();
myEmmiter.on('customEvent',
(data) => { console.log('Event data:
data);
});
```

myEmitter.emit('customEvent', {
message: 'Custom event triggered!'});

7) console:

Use: The console module provides methods to write to output to console or other writable streams.

imp:
console.log :- write message.
→ console.error = "error message
console.time = start timer.
console.timeEnd = stop Timer.

→ program.

```
console.log("Hello world");
console.error("An error.");
console.time("mytimer");
console.timeEnd("mytimer");
```

8) Buffer:

- Use: It is provide a way to handle binary Data in Node.js such as reading from streams or manipulated raw data.

→ imp

Buffer.from: Create New Buffer object from a given input data.

- buf.toString = Decodes and returns the data in the buffer object as a string.

- buf.length: return the Number of bytes in Buffer object.

1. Program:

```
const buf = Buffer.from('Hello',
    'UTF-8');
console.log(buf.toString());
console.log(buf.length);
```

9) querystring:

Use: it module provides utilities to work with TIve streams. which are Used to encode URL parameters.

→ imp

querstring.parse : purse a query string and return an object.

→ querystring.stringify: Convent an object to a query string.

→ program:

```
const querystring = require('querystring');
const ''          =name=John&age=30';
const parseobj = querystring.pure
                 (querystring);

Console.log (parseobj);
```

const formeted string = querystring.
string(Objecttoquery string);

console.log(formutted string);

10) http:

→ Use: it module provides functionality
For creating HTTP server and
making HTTP request.

⇒ imp:

http.createServer: Create http
server instance
- http.server: The class representing
HTTP server.
- http.request: send request to a
server

→ program:

const http = require('http');
const server = http.createServer

```
((req, res) => {
res.writeHead(200, {'content-type':
    'text/html'})
res.end("Hello");
});

server.listen(8000, () => {
    console.log('server Running on
                8080');
});
```

11) V8

→ use: it module exposes APIs to access the V8 JS engine internal functionality, such as memory and cpu profiling.

→ imp:

V8.getheadstatistics(): returns an object with statistics about the V8 heap memory usage.

→ V8.getheapspacestatics(): return an array of object with statistics for each space in the V8 heap.

→ program:

```
const V8 = require('v8');
const heapstats = V8.getheapstatistics
console.log(heapstats);
```

12) OS :-

→ use: it is module provides information about the operating system and related functions.

→ imp:

OS.cpus(): return an array or object containing information about each CPU core

→ os.totalmem(): return the total system memory in bytes.

→ os.freemem(): return the free memory in bytes.

→ program.

```
const os = require('os');
const cpus = os.cpus();
console.log(cpus);
const totalMemory = os.totalmem();
const totalMemory = os.freemem();
console.log('Total memory =" +
    totalmemory + bytes');

console.log(' free Memory : ${freemem
    bytes');
```

13) Zlib:

→ Use: it provide comporision and decompression functions utilities for data stream using CTzip declare and other algorithm.

→ program:

zlib.createZip: create GZip object to compress Data

→ zlib.createGunzip: Create a Gunzip object to Decompress Data.

→ program:

```
const zlib = require('zlib');
const fs = require('fs');

const input = fs.createReadStream
              ("input.text");

const output = fs.createWriteStream
               ("input.txt");

input.pipe(zlib.createGzip()).
           pip(output);
```