Page No.
Date

Name :- Vinadiya Nenshi Hareshbhai

Roll no :- 079

Semester :- 7th sem [M.Sc(IT)]

Subject :- Application Development using Full Stack [701]

[Theory Assignment :- 1]

Date

1] Node.js :- Introduction, features, execution architecture

→ Introduction :-

Node.js is a server-side platform built on Google chrome's JavaScript Engine (v8 Engine).

- Node.js was developed by Ryan Dahl in 2009 & its latest version is v0.10.36.
- Node.js is an open source, cross-platform runtime environment for developing server side & networking applications.
- Node.js applications are written in JS, & can be run within the node.js runtime on OS X, MS windows, & Linux.

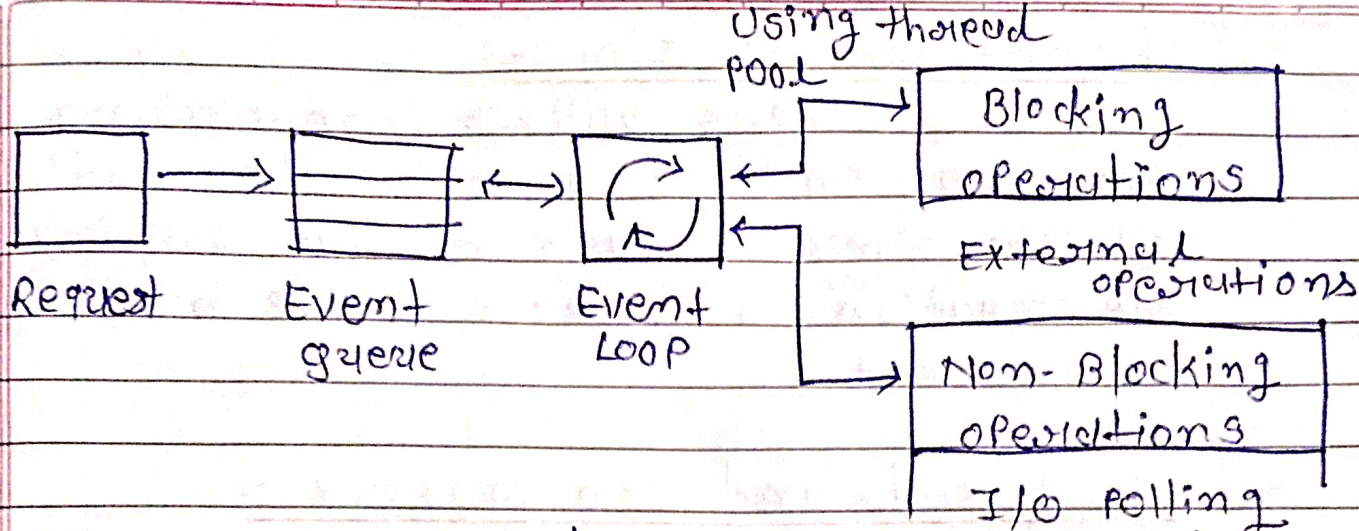
Node.js = Runtime Environment + JS library

→ Features of Node.js :-

1) Asynchronous & Event Driven.
2) very fast
3) single threaded but Highly scalable
4) NO Buffering
5) License.

→ Execution Architecture :-

- Node.js offers a "single-threaded Event Loop" architecture to manage concurrent requests without creating multiple threads & using fewer threads to utilize fewer resources.



- The reason for the popularity of Node.js is its callback mechanism & JS-event-based Model.

- **1] Event loop :-**
The event loop is the core of node asynchronous and non-blocking architecture. It continuously runs & wait for events to occur. when an event is detected, it triggers the corresponding callback function, & the event loop moves on to the next event.

- **2] callBacks :-**
callbacks are a fundamental part of node.js development when asynchronous allowing developers to handle the results or errors accordingly callbacks are passed as arguments to asynchronous functions.

- **3] Event Emitters :-**
Event Emitters are objects in node.js that can emit named events and attach listeners to those events. They form the basis of many core node.js modules & are widely used for handling various asynchronous operation.

Date

- 4] Non-Blocking I/O :-

Node.js utilizes Non-blocking I/O operations, enabling it to handle multiple requests simultaneously without waiting for one operation to complete before moving to the next.

- 5] Thread and concurrency :-

Though node.js runs on a single thread, It employs a thread pool for executing certain I/O operations like file system access.

- 6] Modules and NPM :-

Node.js encourages the use of modules which are independent units of functionality that can be reused across applications. These modules are managed using NPM, which simplifies dependency management & makes it easy to integrate third-party libraries.

2] Note on Modules with example.

→ Modules are fundamental concept in node.js that allows developers to organize code into reusable and maintainable units. In node.js a module is essentially a separate file containing JS code that can be imported & used in other files. This modularity is

a key factor in making Node.js applications managable & scalable.

- Create Modules in Node.js :-

- i) Create a new file :-
start by creating a new file with discriptive name that reflects the purpose of the module. Exa, file name = "mathutils.js" it contains math functions.

- ii) Define the functionality :-
within the 'mathutils.js' files define the functions we want to make available as part of the module. for exa :-

```
function add(a,b)
{
    return (a+b);
}
module.exports = {add};
```

- iii) Export the module :-
To make the function available outside. In this exa, we are exporting the add function.

→ Benefits of using modules :-

→ code organization & reusability :-
Modules allow developers to organize code into logical units. making it easier, to manage & maintain larger

Page No.
Date

Projects by breaking down functionality into modules.

→ Encapsulation :-
Modules provide a level of encapsulation as the internal details of a module are hidden from the outside scope only the functions & variables explicitly exported through module.

→ Maintainability & collaboration :-
The use of modules facilitates collaboration among developers working on the same project different team members can work on separate module independently.

3] Note on Package with example.

→ A package in node.js contains all the files we need for a package module.

- For creating a package :-

Step :- I → Initialize a new node.js project create a new folder for a package & navigate to it into the terminal. command :-

```
npm init
```

Step 2 → create the main code file. create a new file in a project folder.

Page No.
Date

Exa :- function add (a,b)
{
return a+b;
}
module.exports = { add };

Step 3 :- Add Package information
open the package.json file that was generated & add any additional information you want to include such as the package name, description, author, licenses, etc.

Step 4 :- Test the Package locally before publishing a package, let's create a test file.

Exa :-
const myPackage = require('./index');
console.log('Adding 2 and 3 :', myPackage.add(2,3));

Step 5 :- Publishing a Package
If you want to share your package with others, you can publish it to the npm registery.
npm publish

4] Use of Package.json and Package-lock.json

→ Package.json :-
It is JSON file that contains metadata about a node.js project & the list of dependences required for project to run.

- It serves as the configuration file for the project & provides essential information, such as project's name, version, description, entry point, author, license & more.
- Ex:-

```
{
  "name" :- "myProject",
  "version" : "1.0.0",
  "description" : "Project",
  "Main" : "index.js",
  "author" : "JohnDoe",
  "license" : "MIT",
  "dependencies" : {
      "express" : "^4.17.1",
      "lodash" : "^4.17.21"
  }
}
```

→ **Generate package.json :-**

We can generate package.json file by running "npm init" in project's directory.

→ **Package-Lock.json :-**

This file automatically generated by npm when we run 'npm install'.

→ **Use :-**

Dependency version Locking :-

The Package-Lock.json file records the exact version's of all dependencies installed in the 'node_modules' directory.

This means that even if the project's 'package.json' specifies a range of versions for dependencies, the exact version installed on your system will be determined by 'package-lock.json'.

- Example of package-lock.json :-

```
{
  "name" : "NodeProject",
  "lockfileVersion" : 2,
  "requires" : true,
  "dependencies" :
  {
    "express" : {
      "version" : "4.17.1",
      "resolved" : "https://registory/..";
      "integrety" : "sha512 ---- ",
      "dev" : false,
      "requires" : {
          "qs" : "6.7.0"
      }
    },
    "qs" : {
      "version" : "6.7.0",
      "resolved" : "https://registory.npm... ",
      "integrity" : "sha512-vcdBRNFT.... ",
      "dev" : false
    }
  }
}
```

5] Node.js Packages :-

→ Node.js packages are collections of reusable code and modules that can be easily shared installed & integrated into node.js projects.

- A package is simply a directory containing a 'package-json' file that describes the package & its dependencies.

- It may also contain other files, such as JS code, configuration files, documentation, tests, and more.

- Packages are typically published on the npm registory, a public repository for Node.js packages, making them easily accessible to developers world wide.

→ npm - Node package Manager :-

- npm is the default package manager for node.js & stands for 'Node Package Manager'.

- It allows developers to easily discover, install, update & manage Node.js packages.

→ Key features & functionalities of NPM :-

1) Package installation
2) Package management
3) Sementic versioning (semVer)

4) Package Publishing
5) Dependency Management
6) Script
7) Version Tagging
8) Scoped Packages
9) Security and Auditing.

npm has revolutionized the node.js ecosystem by providing a seamless & efficient way to manage packages & dependencies.

6] npm introduction & commands with its use.

→ npm (node Package Manager) is a package manager for node.js, allowing developers to easily install, manage & update packages of reusable code.

- It simplifies the process of adding external libraries, modules & tools to a Node.js project.

- 1) npm install :-
npm comes bundled with node.js so when you install node.js on your system, npm is automatically installed alongside it.

- 2) Package.json :-
This file is the heart of any node.js project. It contains metadata about the project.

- 3) installing packages :-
To install a package, use the npm install cmd followed by package name.
- exy :-
npm install lodash

- 4) Depencies :-
when you install package using npm, it automatically adds the package as a dependency in the package.json file.

- 5) uninstalling Packages :-
To remove package from your project use the 'npm uninstall' command followed by Package name.
- Exa :-
npm uninstall Lodash

- 6) update packages :-
To update packages to the latest version, you can use the 'npm update' command followed by the package name.

- 7) Free search for Packages :-
you can search for Packages on the npm registery using 'npm search' followed by a keyword.
- exa :-
npm search logging

npm plays a crucial role in the node.js making it easier for developers to manage dependences & share code.

7] Describe use & working of following node.js Packages important properties & methods & relevent programs.

url
process, Prml (External Package)
readline
fs
event
console
buffer
querystring
http
vs
os
zlib

→ I) url :-

- use :- The url Package provides utilities for URL resolution & Parsing.

- working :- It allows you to manipulate URLs; Parse their component and resolve relative URLs into absolute URLs.

- Properties & methods :-
  url.Parse
  url.format
  url.resolve

- Exa :-
```
const url = require('url');
const urlString = "www.google.com";
```

```
const ParseUrl = url.parse(urlString, true);
console.log(ParseUrl);

const formatUrl = url.format(ParseUrl);
console.log(formatUrl);
```

→ 2] Process :-

- Use :- The Process object provides information & control over the current node.js Process.

- Working :- It allows interaction with the Process, including listening for signals, accessing env variables, terminating processes.

Properties & Methods :-

Process.argv
Process.env
Process.exit(code)

- Ex :-

```
console.log(Process.argv);
console.log("first argument", Process.argv[2]);
```

→ 3] readline :-

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

- working :- It allows you to read data line by line making it useful for building cmdline interfaces.

Properties & methods :-
readline.createInterface
rl.on('line', callback)
rl.close();

- Eg :-

```
const readline = require('readline');
const rl = readline.createInterface({
    process.stdin, process.stdout});

rl.question('what is your name', (name)=>
{
    console.log('Hello, ${name}');
    rl.close();
});
```

→ 4] fs :-

- Use :- The fs module provides file system relate functionality, allowing you to read from & write to files.

- Working :- It interacts with the file system, performing operations like reading, writing, deleting, renaming files.

Properties & methods :-
fs.readfile
fs.writefile
fs.unlink

- Eg :-

```
const fs = require('fs');
fs.writefile('myfile.txt', 'Hello', (err) => {
    if(err)
    {
        console.error(err);
}
```

```
    else {
        console.log("written");
    }
});
```

→ 5] events :-

- Use :- The events module provides an event-driven architecture for building scalable application with event emitters & listeners.
- working :- It allows you to create custom events & register event listners to handle those events.

Properties & methods :-

events.Emitter
eventEmitter.on(e_name, listner)
eventEmitter.emit(e_name, [args])

- Ex :-

```
const event = require('events');
class myemit extends EventEmitter {}

const myemitter = new myEmitter();
myemitter.on('greet', (name) => {
    console.log("Hello", ${name}!);
});

myemit.emit('greet', 'John');
```

→ 6] console :-

- Use :- The console module provides a simple debugging console similar to the browser's console object.

- Working :- It allows you to log information, errors & writing during the development phase.
- Methods :-
  console.log()
  console.warn()
  console.error()
- Exy :-

```
console.log("Hello");
```

→ 7] Buffer :-
- Use :- The Buffer module provides a way to handle binary data.

- Working :- It allows you to work with streams of binary data, such as reading & writing files.

  Property & methods :-
  Buffer.alloc(size)
  Buffer.from(data)
  Buf.toString([encoding])
- Exy :-

```
const bufI = Buffer.alloc(8);
console.log(bufI);
```

→ 8] QueryString :-
- Use :- This module provides utilities for Parsing & formatting URL query strings.

- Working :- It allows you to work with the query component of a URL, which is commonly used to pass data in HTTP requests & responses.

Property & Methods :-

queryString.parse(str)
queryString.stringify(obj)

- Exp :-

```
const qs = require('queryString');
const qs = 'name = Nenshi';
const pq = queryString.parse(qs);
console.log(pq);
```

→ 9] http :-

- Use :- The http module provides functionality for http server & making HTTP requests.

- working :- It allows you to build web servers, handle incoming requests, & send HTTP request to other servers.

Properties & Methods :-

http.createServer()
http.get(url, callback);
Server.listen(port, [host], callback)

- Exp :-

```
const http = require('http');
const server = http.createServer((req, res) =>
    res.end('Hello');
});
server.listen(8000, () => {
    console.log("listening");
});
```

→ 10) V8 :-

- Use :- The V8 module provides access to the V8 JS engine's Internal features.

- Working :- It allows you to gather information about memory use, CPU usage, other low-level V8 Engine information.

Properties & Methods :-

```
V8.getHeapStatistics();
V8.cachedDataVersionTag();
```

- Eg :-

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

→ 11) OS :-

- Use :- The OS module provides OS related utility functions.

- Working :- It allows you to access information about the OS, such as network interface, CPU architecture & system uptime.

Properties & methods :-

```
os.platform()
os.cpus()
os.totalmem()
```

- Eg :-

```
const os = require('os');
console.log('Platform:', os.platform());
console.log('CPU:', os.cpus());
console.log('Memory:', os.totalmem());
```

→ 19) zlib :-

- Use :- The zlib module provides compression & decompression functionalities using GZip, Deflate, & Brotli algorithms.

- Working :- It allows you to compress data for efficient storage & transmission & decompress it when needed.

Properties & Methods :-

```
zlib.gzip(input, callback);
zlib.gunzip(input, callback);
zlib.deflate(input, callback);
```

- Exp :-

```
const zlib = require('zlib');
const data = 'Hello N';

zlib.gzip(data, (err, compressedData) => {
    if (err) {
        console.error(err);
    }
    else {
        console.log(compressedData);

        zlib.gunzip(compressedData, (err, ddata) => {
            if (err) {
                console.error(err);
            } else {
                console.log(ddata.toString());
            }
        });
    }
});
```