Theory Assignment 1

Name :-
Katrodiya Hasti Atulbhai

Roll No. :-
022 [Twenty-Two]

Subject :-
Application Development using Full Stack.

Sem :-
M.Sc IT 7th sem

Date

Date
Page

Que :- Node Js :- Introduction, features, execution Architecture.
[1]

Ans :- Node.js is an open-source, cross-platform, server-side javascript runtime environment that allows developers to build scalable and efficient network applications.

→ It was first introduced in 2009 by Ryan Dahl and has since gained immense popularity in the web development community.

→ Node.js enables developers to run JavaScript code outside of the browser, making it suitable for building server-side applications and real-time web application.

Features of Node.js

[1] Asynchronous and None-Blocking I/O :-
Node.js is built on the principles of non-blocking, even-driven architecture. It employs an event loop that allows asynchron--ous processing of I/O operation. This means that node.js application can handle multiple request concurrently without getting blocked, making it highly efficient and suitable for handling a large number of concurrent connection.

[2] Single - Threaded and Event - Driven :-

Node.js runs on a single threaded event loop which helps in handling concurrent request efficiently, while traditional server side technologies create seperate threads for each request, Node.js uses a single thread to handle multiple requests. When a request is received, it triggers an event and the event loop manages the asynchronous I/O operation.

[3] Fast Execution :-

Node.js is built on Google V8 Javascript engine which compiles Javascript code into native machine code. This allows node.js applications to execute with high speed and efficiency. V8 engine is also used in google chrome browser, which ensures a consistent and high - performance execution environment for Javascript.

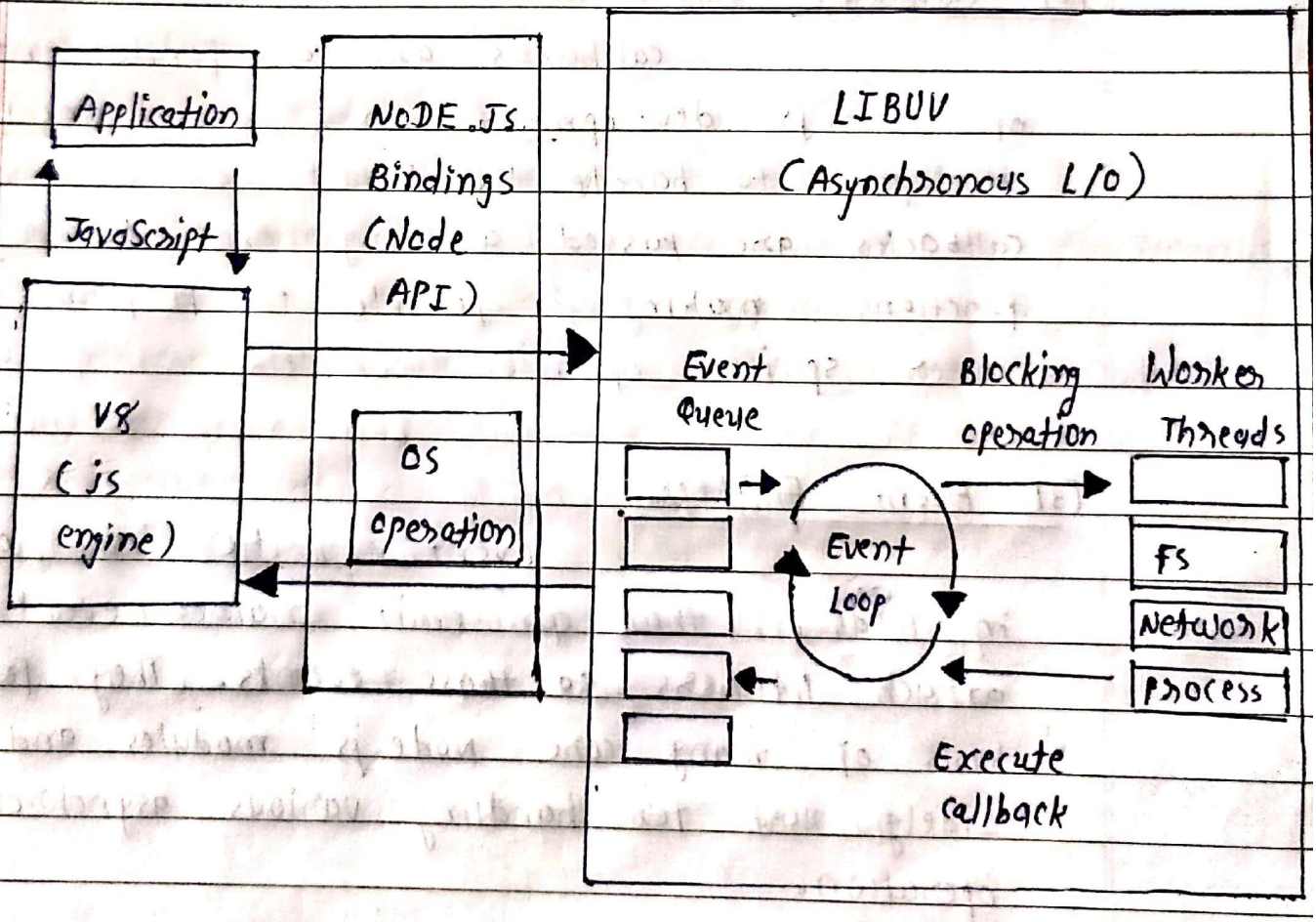
[4] NPM (Node Package Manager) :-

Node.js comes ~~ther~~ with a powerful package manager called NPM, which allows developers to install, publish and manage packages and dependencies easily. NPM hosts a vast repository of open - source package that can be used to extend the functionality of Node.js applications. This extensive ecosystem of libraries and modules significantly accelerates the development process.

[5] Extensible and Modular :-

Node.js encourages a modular approach to development, allowing developers to break down application into smaller, reusable modules. This modularity fosters code reusability, maintainability and scalability, making it easier to manage large codebases.

Execution Architecture of Node.js



Date
Page

[1] Event Loop :-

The event loop is the core of node asynchronous and non-blocking architecture. It continously runs and wait for events to occur. When an event is detected, it triggers the corresponding callback function, and the event loop moves on to the next event. This loop allows Node.js to handle multiple requests concurrently without blocking the execution of other code.

[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, making it possible to perform action after specific operation complete.

[3] Event Emitters :-

Event emmiters 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 and are widely used for handling various asynchronous operation.

[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. This approach makes Node.js highly efficient, especially in applications that require frequent I/O operations.

[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. This allows Node.js to take ~~acknowledge~~ advantages of multicore processors while still maintaining its single-threaded, event-driven nature.

[6] Modules and NPM :-

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

Que :- [2] Note on Modules with example.

Ans :- Modules are fundamental concept in node. is that allows developers to organize code into reusable and maintainable units. In node.js a module is essentially a seperate file containing Javascript code that can be imported and used in other files. This modularity is a key factor in other files. making Node.js applications manag- -able and scalable.

Create Modules in Node.js

To create a module in node.js,

i) Create a new file :-

start by creating a new file with discriptive name that reflects the purpose of the module. for example, let's create a file called "mathutils.js" that will contain some basic 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 example -

```
function add(a, b)
{
    return (a+b);
}

module.exports = { add };
```

iii) Export the module :-

To make the function available outside. In this example we are exporting the add function.

Benefits of using modules :-

→ Code Organization and reusability :-

Modules allow developers to organize code into logical units, making it easier, to manage and maintain larger projects by breaking down functionality into modules. code reusability is promoted reducing the need for redundant code.

→ Encapsulation :-

Modules provide a level of encapsulation as the internal details of a module are hidden from the outside scope only the functions and variables explicitly exported through 'module. exports' are accessible externally. This helps in preventing unintended modifications to module internals.

→ Maintainability and collaboration :-

The use of modules facilitates collaboration among developers working on the same project different team members can work on separate module independently and as long as the module interface remains the

consistent, changes in one module are less likely to affect others.

Que:3 Note on Package with example.

Ans :- A package in Node.js contains all the files we need for a package module.

→ for creating a package :-

Step 1 :-

Initialize a new node.js project create a new folder for a package and navigate to it into the terminal. Then, initialize a new Node.js project by running this command :-

```
npm init
```

Step 2 :-

create the main code file. create a new file in a project folder

ex:-

```
// index.js
function add (a, b)
{
    return a+b;
}

module.exports = { add };
```

Step 3 :- Add package information.

open the package.json file that was generated and 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.

ex :-

test.js

```
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 registry

```
npm publish
```

Que :-
[4] Use of package.json and package-lock.json

Ans :- **Package.json :-**

package.json is JSON file that contains metadata about a node.js project and the list of dependences required for project to run.

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

ex :-

```
{
  "name" : "myProject",
  "version" : "1.0.0",
  "description" : "project",
  "main" : "index.js",
  "author" : "John Doe",
  "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 :-

The package-lock.json file is automatically generated by npm when we run 'npm install'.

→ It is used to lock the versions of dependencies to ensure consistent and reproducible installations across different environments.

Use :-

Dependency version Locking :-

The package-lock.json file records the exact versions 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'.

→ This will be determined by 'package-lock.json'. This ensures that everyone working on the project uses the same versions of dependencies, preventing potential issues caused by different versions.

example of package-lock.json

```
{
    "name" : "NodeProject",
    "lockfileVersion" : 2,
    "requires" : true,
    "dependencies" :
```

```
{
  "express": {
    "version": "4.17.1",
    "resolved": "https://registry...",
    "integrity": "sha512-...",
    "dev": false,
    "requires": {
      "qs": "6.7.0"
    }
  },
  "qs": {
    "version": "6.7.0",
    "resolved": "https://registry.npm...",
    "integrity": "sha512-VCdBRNFT...",
    "dev": false
  }
}
```

→ In conclusion, package.json and package-lock.json files work together to manage project metadata and dependency versions effectively.

Date
Page

Que:-
[5] Node.js Packages.

Ans:-

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

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

→ It may also contain other files, such as JavaScript 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 worldwide.

npm - Node Package Manager

→ npm is the default package manager for node.js and stands for 'Node Package Manager.'.

→ It allows developers to easily discover, install, update and manage Node.js packages.

→ npm connects to the npm registry by default, where thousands of opensource packages are hosted.

Key features and functionalities of NPM includes :-

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 and efficient way to manage packages and dependencies.

Que :-
[6] npm introduction and commands with its use.

Ans :- npm (node package Manager) is a package manager for node.js, allowing developers to easily install, manage and update packages of reusable code.

→ It simplifies the process of adding external libraries, modules and 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 :-
The package.json 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.
ex :-
npm install lodash

4) Dependencies :-
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.

ex :-

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) Execute scripts :-

In the package.json file, you can define script that can be executed using the 'npm run' command.

You can define a script called start to run your application, then use 'npm run start' to start your node.js application.

8) Search for packages :-

you can serch for packages on the npm registery using 'npm search' followed by a keyword.

ex :- npm search logging

9) Publishing packages :-

if you have developed node.js module or library and want to share it with the community, you can publish it to the npm registry using the 'npm publish' command.

npm plays a crucial role in the node.js making it easier for developers to manage dependences and share code, ~~for~~ fostering collaboration and code reuse among the node.js community.

Que :-

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

url
process, pm2 (External package)
readline
fs
event
console
buffer
querystring
http
v8
os
zlib

i) url :-

Use :- The url package provides utilities for URL resolution and 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

ex :-

```
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);
```

ii) 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]);
```

iii) 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 interface.

properties & methods :-

readline.createInterface
rl.on('line', callback)
rl.close();

ex :-

```
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();
});
```

iv) fs :-

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

working :-

It interacts with the file system, performing operations like reading, writing, deting, renaming files.

Properties & Methods :-

fs.readfile
fs.writefile
fs.unlink

ex :-

```
const fs = require('fs');
fs.writeFile('myfile.txt', 'Hello', (err)=>{
    if (err)
    {
        console.error(err);
    }
    else {
        console.log("written");
    }
});
```

v) events :-

Use :- The events module provides an event-driven architecture for building scalable applications with event emitters & listeners.

working :-

It allows you to create custom events & register event listners to handle those events.

Properties & Methods

events.EmitterS
eventEmiter.on(eventname, listner).
eventEmitter.emit(eventname, [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');
```

vi) 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 and writing during the development phase.

Methods :-

console.log()
console.warn()
console.error()

ex :-

```
Console.log("Hello");
console.warn("warning");
console.error("errors");
```

vii) 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, network communication etc.

Property & methods :-

Buffer.alloc(size)
Buffer.from(data)
buf.tostring([encoding])

exc :-

```
const buf1 = Buffer.alloc(8);
console.log(buf1);

const buf2 = Buffer.from('Hello');
console.log(buf2);

const buf3 = buf2.toString('utf8');
console.log(buf3);
```

viii) QueryString :-

Use :- This module provides utilities for parsing and 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 and responses.

Methods :-

querystring.parse(str)
querystring.stringify(obj)

ex :-

```
const qs = require('querystring');
const qS = 'name=hasti';
const pq = querystring.parse(qS);
console.log(pq);

const obj = {name: 'Hasti'};
const FQ = querystring.stingify(obj);
console.log(FQ);
```

IX) http :-

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

Working :-

It allows you to build web servers, handle incoming requests, and send HTTP request to other servers.

Properties & methods :-

http.create server().
http server.listen(port, [host], callback)
http.get(url, callback);

ex :-

```
const http = require('http');
const server = http.createServer((req, res) =>
    res.end('Hello');
});

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

x) v8 :-

Use :- The v8 module provides access to the v8 JavaScript 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.cachedataVersionTag();
```

ex :-

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

xi) os :-

Use :- The os module provides operating system related utility functions.

working :-

It allows you to access information about the os, such as network interface, CPU architecture and system uptime.

properties & Methods :-

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

ex :-

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

xii) zlib :-

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

working :-

It allows you to compress data for efficient storage and transmission and decompress it when needed.

properties & Methods :-

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

ex :-

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

const data = 'Hello H';
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());
      }
    });
  }
});
```

These node.js packages provides essential functionalities to help developers build robust and featurich applications.