Theory Assignment - 1

Name: Sakshi Umeshkumar Maisuria

Semester : 7th

Roll NO : 26.

Subject : Full Stack Development

Code : 701.

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

Ans-1. • Introduction to node.js :

- Node.js is an open-source and cross-platform runtime environment for executing javascript code outside a browser.
- We need to remember that NodeJs is not a framework and it's not a programming language.
- We often use node.js for building back-end services like APIs like web App or Mobile App.
- It's used ~~to~~ in production by large companies such as Paypal, Uber, Netflix, Walmart and so on.

node.js = Runtime Environment + JavaScript Library.

- Node.js is free,
- Node.js runs on various platforms like windows, Linux, Unix, Mac os x, etc.
- It uses javascript on the server
- Node.js uses asynchronous programming.

• Features of Node.js :

- There are other programming Languages also which we can use to build back-end services so what makes Node.js different.

1. It's ~~can going~~ easy to get started and can be used for prototyping and agile development.

2. It provides fast and highly scalable services.
3. It uses javascript everywhere, so it's easy for a Javascript programmer to build back-end services using Node.js.
4. Source code cleaner and consistent.
5. Large ecosystem for open source library
6. It has Asynchronous or Non-blocking nature.

- Execution Architecture:

- Node.js uses a Single Threaded Event loop architecture to handle multiple concurrent clients.
- The node.js processing model is based on a javascript-event-based model and a callback mechanism. This mechanism based on the event loop allows node.js to run blocking I/O operations in a non-blocking way.
- So, It employs two fundamental concepts:
  1) Asynchronous model.
  2) Non-blocking of I/O operations.
- This features enhance the scalability, performance and throughput of node.js Web applications.
- The node.js architecture consists of the following components:

i) Requests:
- Depending on the actions that a user needs to perform, the question requests to the server can be either blocking (complex) or non blocking (simple).

ii] Node.js server:
- The node.js server accepts user requests, processes them and returns results to the users.

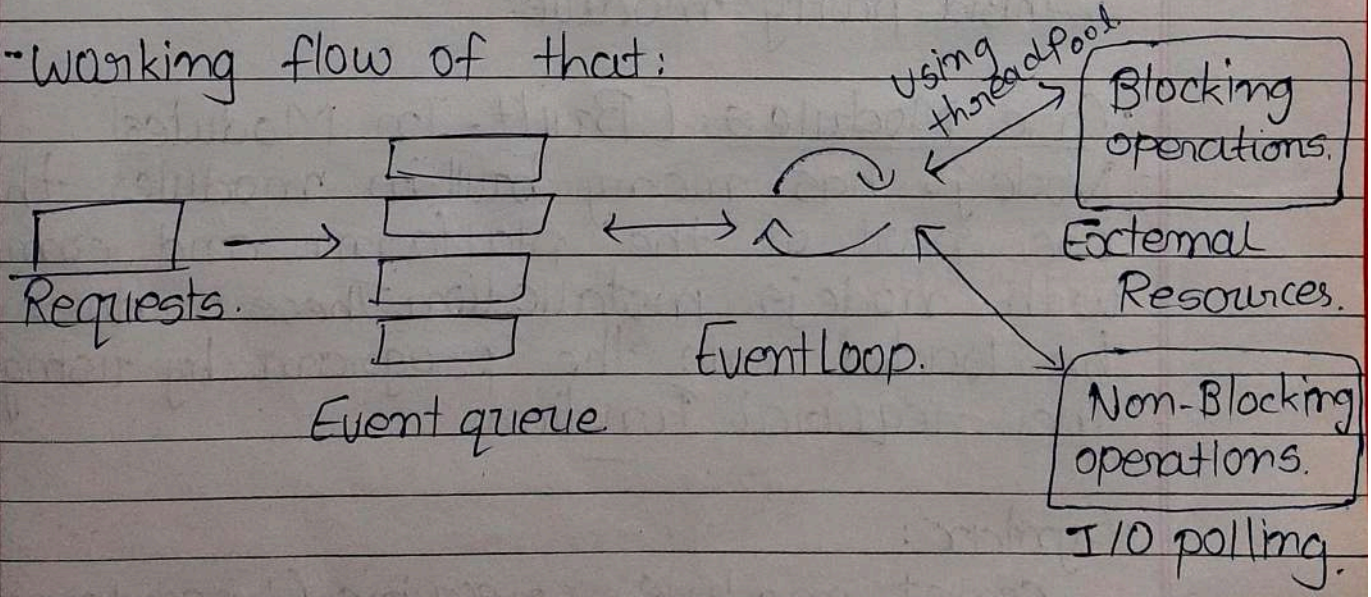
iii] Event queue:
- The main use of Event queue is to store the incoming client requests and pass them sequentially to the Event Loop.

iv] Thread Pool:
- The thread pool in a node.js server contains the threads that are available for performing operations required to process requests.

v] Event loop:
- Event loop receives requests from the Event queue and sends out the responses to the clients.

vi] External Resources:
- In order to handle blocking client requests, external resources are used. They can be of any type. (computation, storage, etc).

- Working flow of that:



Que-2. Note on modules with Eacample.

Ans-2. · Modules:

- A modules to be the same as Javascript libraries.
- In node.js, Modules are the blocks of encapsulated code that communicate with an external application on the basis of their related functionality.
- Modules can be a single file or a collection of multiple, files / folders.
- The reason programmers are heavily reliant on module is because of their reusability as well as the ability to break down a complex piece of code into manageable chunks.

- In node.js, there are three types of modules.
  - Core module [built-in]
  - local module.
  - Third-party module.

· Core Modules: [Built-In Modules]

- Node.js has many built-in modules that are part of the platform and come with node.js installation. These modules can be loaded into the program by using an the required function.

· Syntax:

```
const module = require('module name');
```

- The require() function will return a Javascript type depending on what the particular module returns.

example: create to a web server using http module:

```
const http = require('http');
http.createServer(function(req, res){
    res.writeHead(200, {'Content-type': 'text/HTML'});
    res.write('Welcome module');
    res.end();
}).listen(8000);
```

- In the above example, the require() function returns an object because the http module returns its functionality as an object.
- The http.createServer() method will be executed when someone tries to access the computer on port 8000. The res.writeHead() method is the status code where 200 means it is ok, while the second argument is an object containing the response headers.

- Core Modules:	Description:
http.	creates an HTTP server in node.js
assert.	set of assertion functions useful for testing.
fs.	used to handle file system.
path.	includes methods to deal with file paths.
process.	provides information and control about the current node.js process.

os. provides information about the operation system.

querystring utility used for parsing & formatting URL query string.

url: module provides utilities for URL resolution and parsing.

• Local Modules:

- Unlike built-in and external modules, local modules are created locally in your node.js application.

ex: create a calc.js file:

```
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 file:

```
consde
const calculator = require('./calc');
let x=50, y=10;
console.log("Addition :" + calculator.add(x,y));
console.log("Substraction:" + calculator.sub(x,y));
console.log("Multiplication:" + calculator.mult(x,y));
```

```
console.log("Division:" + calculator.div(x, y));
```

Output: Addition : 60
Substraction : 40
Multiplication : 500
Division : 5.

• Third-Party Modules:

- Third-party modules are modules that are available online using the node package Manager. (NPM) Thiese modules can be installed in the project folder or globally. Some of the popular third-party modules are Mongoose, express, angular, React.

ex:

```
npm install express
npm install mongoose
npm install -g @angular/cli.
```

Que-3 Note on Package with Example.

Ans-3 • Package:

- A package in node.js contains all the files you need for a module.
- Modules are javascript libraries you can include in your project.
- A package is a way to organize related classes, interfaces and resources together in a structured manner.
- It provides a mechanism for creating namespaces and prevents naming conflicts, making it easier to manage and maintain large codebases.
- Packages are commonly used in programming languages like Java, Python & C#.

• Download a Package:
- It's very easy. Open the command line interface and tell NPM to download the package you want.

ex-

```
>npm install upper-case
```

- npm creates a folder named "node_modules", where the package will be placed.

• Using the Package:
- Once the package is installed, it is ready to use.

ex-

```
var uc = require('upper-case');
```

ex: create a file that will convert the output into upper-case letters:

```
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);
```

Output: HELLO WORLD.

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

Ans-4] · Package.json :

- The package.json file is the heart of any Node project. It records important metadata about a project which is required before publishing to NPM, also defines functional attributes of a project that npm uses to install dependencies, run scripts and identify the entry point to our package.

- The main field is a module ID that is the primary entry point to your program.
- The 'package.json' file is a metadata file for node.js project. It categories info below categories:
  i] Identifying metadata properties.
  ii] Functionall metadata properties.
- It serves multiple purposes, including:

· Dependency Management:
It lists all the direct dependencies (and their versions) required by your project. These dependency can be packages from NPM or other sources. When someone else wants to run your project or when you deploy it to a production server, they can use the 'package.json' to install all the required dependencies automatically.

· Project Information: It includes details about project such as its name, description, version, author, license, entry point (the main file of your application), scripts and more

· Script Definition:
It allows you to define custom scripts that can be run using the 'npm run' command. For example, you can define scripts to start your applications, run tests, build assests and more.

· Dependency Version Control:
- It enables developers to define version ranges for dependencies. These version ranges can be exact versions, minimum versions, or flexible ranges. This helps in maintaining compatibility between different packages used in this project.

· A demo package.json file with the required information.

```
{
  "name" : "GeeksForGeeks",
  "version" : "1.0.0",
"description" : "GFG",
"main" : "index.js",
"scripts" : {
    "test": "echo \"Error: no test specified \" && exit 1",
    "start": "node start.js",
},
"engines" : {
      "node" : ">=7.6.0",
      "npm" : ">=4.1.2"
},

"author" : "GFG",
"license" : "ISC",
```

```
"dependencies" : {
    "body-parser" : "^1.17.1",
    "express" : "^4.15.2",
    "express-validator" : "^3.1.2",
    "mongoose" : "^4.8.7",
    "nodemon" : "^1.14.12",
},
"devDependencies" : { },
"repository" : {
        "type" : "git",
        "url" : "https://github.com/gfg/gfg.git"
},
"bugs" : { "url" : "https://github.com/gfg/gfg/issue" },
"homepage" : "https://github.com/gfg/gfg#readme"
}
```

- Package-lock.json:

- package-lock.json is automatically generated for any project operations where npm modifies either the node_modules tree, or package.json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

- This file is intended to be committed into source repositories, serves various purposes:

- Describe a single representation of a dependency tree such that teammates, deployments and continuous integration are guaranteed to install exactly the same dependencies.

- Provide a facility for user to "time-travel" to previous states of node_modules without having to commit the directly itself.

- Facilitate greater visibility of tree changes through readble source control diffs.

- Optimize the installation process by allowing npm to skip repeated metadata resolutions for previously-installed packages.

- Fasten Installations: By having the exact version of dependencies listed in the 'package-lock.json' NPM can skip the process of calculating depedency versions during installation. This makes installations faster and reduces the risks of discrepancies between environments.

- So, package.json is used for project metadata and dependency management, while 'package-lock.json' is ~~use~~ ensures a consistent & deterministic installation of dependencies across different environments. Together, these files help maintain the stability and reliability of node.js project.

Que-5] Nodejs Packages:

Ans-5] Nodejs packages are collections of code, modules, or libraries that extends the functionality of nodejs applications.
- These packages can be easily installed and managed using the node Packages Manager (NPM), making it convenient for developers to reuse code, enhance their projects, and collaborate with other developers.
- These are two main types of Nodejs packages:

1] Core Packages:
- core packages are built-in modules that come bundled with the nodejs installation.
- They provide essential functionality to interact with the underlying operating system, file system, network and more.
- Developers can use core packages directly in their applications without the need for additional installations.

- ex- of core packages include:

- `fs`:
Provides file system-related functionality like reading, writing, manipulating files.

- `http`:
Enables building HTTP servers and making HTTP requests.

- `path`:
Hepl Helps manipulate file paths.

- events:
Provides event-driven programming capabilities.

- Third-Party Packages:

- Third-party packages are developed by the Node.js community or other organizations. These packages cover a wide range of functionalities, from web frameworks and database drivers to utility libraries and integrations with various APIs.
- Developers can easily install third-party packages via NPM and use them in their projects to save time and efforts.

- To use a third-party package in a node.js project, you typically start by creating a `package.json` file to manage your project's dependencies. Then, you install the desired packages using NPM, and they are listed in the 'dependencies' section of the 'package.json' file.
- For example, to install the popular Express web framework and add it as a dependencies in your project:

```
npm install express
npm install Lodash
npm install Axios
```

- express.js : A fast, unopinionated and minimalist web framework for building web applications and APIs.

- Lodash: A utility library that provides a set of useful functions for working with arrays, objects, string and more.
- Axios: A popular HTTP client library for making HTTP requests.

- So, various packages in node.js are; express, Async.js, Browserify, Lodash, Moment.js, Axios, JSHint, morgan, karma, etc
- By leveraging both core packages and third-party packages, Node.js developers can build powerful, scalable, feature-rich applications with ease and efficiency.

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

Ans:-6] - NPM stands for Node Package Manager.
- npm is the default package manager for node.js, and it's plays a crucial role in the node.js ecosystem by simplifying the installation, management and sharing of node.js packages.
- NPM allows developers to discover and use thousands of third-party packages, as well as manage their project's dependencies and scripts easily.

- npm use:
- It enables you to install libraries, frameworks and other development tools for your project. similar to installing a mobile application from an app store.
- You gain access to safe node.js projects for development.
- It helps you speed up the development phase by using prebuilt dependencies.
- npm has a wide variety of tools to choose from for a no cost.
- Using npm commands doesn't require a lot of learning, as they're easy to understand and make sr use of.
- The command line Interface (CLI) for npm is used to run various commands like installing and uninstalling packages, check npm version, run packages scripts, create package.json file.

• commands with its use:

i) npm install:
- this command is used to install packages. You can either install packages globally or locally. When a package is installed globally, we can make use of the package's functionality from any directory in our computer.
- In the other hand, if we install a package locally, we can only make use of it in the directory where it was installed. So no other folder or file in our computer can use the package.

ex -

npm i <package>
npm install <package>
npm install express, npm -v; npm install -g nodemon.

ii) npm init:
- This is used to create the package.json file.
ex - npm init

iii) npm ls:
- This is used to list the packages or modules in your project together with their dependencies.

iv) npm uninstall:
- This is used to uninstall a package.
ex - npm un <package>
npm uninstall <package>
npm uninstall express.

v) npm publish:
- This is used to publish a package.

vi) npm run.
- This lists the available scripts that can be run on

your project.

vii) npm stop:
- This is used to stop a script.

viii) npm cache:
- This force cleans the npm cache.
ex- npm cache clean --force.

~~viii)~~ ix) npm search:
- This is used to search for a package in your project folder.
ex- npm search <packages>

x) npm view:
- This is used to show data about a package and print it to the screen.
ex- npm view <package>.

xi) npm update:
- Use this command to update an npm package to its latest version.

xii) npm restart:
- Used to restart a package. You can use this command when you'd want to stop and restart a particular project.

xiii) npm start:
- used to start a package when required.

xiv) npm version:
- Show you the current npm version installed on your comp

Que-7/8 Describe use and working of following nodejs packages also, important properties and methods and relevant programs.

Ans-7/8. · Url :

· use:

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

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

- The 'url' module provides utilities for URLs, result resolution and parsing. It can be used to manipulate URLs and extract their components.

· Important Properties:

'url.parse(urlString[, parseQueryString[, slashesDenoteHost[]]]':
parses a url string and returns an object with URL component:

ex- const urlString = 'https://www.ex.com:8000/path';
const parsedUrl = url.parse(urlString, true);
console.log(parsedUrl);

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

var q = url.parse(adr, true);

ex:

```
const url = require('url');
const urlString = 'https://www.ex.com:8000/path';
const parsedUrl = url.parse(urlString, true);

console.log(parsedUrl);
```

→ Process, pm2 (external package):

- Process:

- Use: The process module provides information and control over the current node.js process

- Properties:

process.argv: An array containing command-line arguments.

Process.env: An Object containing the environment variables.

process.exit([code]): Exists the current Process with an optional exit code

arch, pid, platform, release, version, versions etc.

ex:

```
console.log(process.argv);
console.log('Process Architecture: ${process.arch}');
console.log('Process PID: ${process.pid}');
console.log('Process Version: ${process.version}');
```

Date
Page

• Methods of Process:

cwd(): returns path of current workingdirectory.
hrtime(): returns the current high-resolution real time in a [seconds, nanoseconds] array.
memoryUsage(): returns an object having information of memory usage.
process.kill(pid[, signal]): is used to kill the given pid.
uptime(): returns the node.js process uptime in seconds.

ex:

```
Console.log(`Current Directory: ${process.cwd()}`);
Console.log('Uptime: ${process.uptime()}');
```

⇒ PM2 (external package):

• Use:

- Restarting after crashes:
- PM2 allows us to keep processes running endlessly, until we shut down our system.

- Monitoring and managing processes remotely:
Web portals allows you to keep track of remote processes and manage them at any time.

- Wide support:
PM2 isn't limited to just to node.js processes, it can be used to manage NGINX servers, Apache servers etc.

. Restart - Persistence : PM2 remember all your processes and restart them after a system boot.

ex:

```
const http = require("http");
const hostname = "192.162.93.11";
const port = 8000;
const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader("Content-Type", "text/plain");
    res.end("This is the main App");
});
server.listen(port, hostname, () => {
    console.log("server: //${hostname}:${port}/");
});
```

⇒ ReadLine :

• Use :

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

Syntax:

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

• Properties and Methods:

clearLine() : Clears the current line of the specified stream.

clearScreenDown() : Clears the specified stream from the current cursor down position.

createInterface(): Creates an InterfaceObject
cursorTo(): Moves the cursor to the specified position.
emitKeypressEvents(): Fires Keypress events for the specified stream
moveCursor(): Moves the cursor to a new position, relative to the current position.

ex:

```
var readline = require('readline');
var fs = require('fs');
var myInterface = readline.createInterface({
  input: fs.createReadStream('demo1.html')
});
var lineno = 0;
myInterface.on('line', function (line) {
    lineno++;
    console.log('Line number' + lineno + ':'
                +line);
});
```

⇒ fs:

- Use:
The nodejs file system module allows you to work with the file system on your computer.
- common use of fs are:
read files, create files, update files, delete files, rename files.

properties & Methods of fs:

- fs.readfile(): used to read files on your computer.
- fs.appendfile(): appends specified content to a file. If the file doesnot exists, the file will be created.
- fs.open(): task takes a flag as the second argument, if the flag is "w" for writing, the file isopened for writing
- fs.writefile(): replace the specified file and content if it exists.
- fs.unlink(): deletes the specified file.
- fs.rename(): renames the specified file.

ex:

```
const fs = require('fs');
fs.readfile('ex.txt', 'utf8', (err, data) => {
    if(err) throw err;
    console.log(data);
});

const content = 'This is write";
fs.writefile("output.txt',content, (err)=> {
    if(err) throw err;
    console.log('Data written');
});
```

=> Events:

• Use:

The event module provides an implementation of the event, allowing objects to emit and listen for events.

- you can create-, fire-, listen for- your events
- To include the built-in Events module use the require() method.
- In addition, all event properties & Methods are an instance of an EventEmitter object. To be able to access these properties and methods,

ex-

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

var myEventHandler = function() {
    console.log('I am');
}
eventEmitter.on('scream', myEventHandler);
event.Emitter.emit('scream');
```

=> Console:

· Use:

- 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 & error message.
- It is used in synchronous way when the destination is a file or a terminal and in asynchronous way when the destination is a pipe

- Methods & Properties:

- Console.log([data],[,...])
- Console.info([data][,...])
- Console.error([data][,...])
- console.warn([data][,...])
- console.dir([data][option])
- console.time([data][label])
- Console.timeEnd([data][label])
- Console.trace([data][,...])
- Console.assert(value[,message][,...])

ex: console.info("Program Started");
var counter = 10;
console.log("counter: %d", counter);
console.time("Getting Data");
// process
Console.timeEnd("Getting Data");
Console.info("program Ended");

⇒ Buffer:

Use:

- The buffers module provides a way of handling streams of binary data.
- The buffer object is a global object in node.js, it is necessary to impact it using the require method.

- Properties & Methods:

alloc(), concat(), from(), includes(), indexOf(), compare(), isBuffer() etc.

ex:

```
var Buf = Buffer.from('abc');
console.log (Buf);
```

- Creating an empty buffer:

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

⇒ QueryString:

- Use: module provides a way of parsing the URL query string.

- Methods & Properties:

escape(): returns an escaped querystring
parse(): Parses the querystring & returns an object.

stringify(): Stringifies the object, returns a querystring.
unescape(): Returns an unescaped querystring.

ex:

```
var qs = require("querystring");
var q = qs.parse("year=2017");
console.log(q.year);
```

⇒ http:

· Use: It allows creating HTTP servers & making HTTP requests. also allows node.js to transfer data over the HTTP.

· Methods & Properties:

'http.createServer([options][, requestListener])'
create an http server instance.

· http.get(options[, callback]): Performs an http GET request.

ex:

```
var http = require('http');
const server = http.createServer((req, res)=>{
  res.writeHead(200,{'Content-Type':'text/plain'});
  res.end('Hello, Node.js!');
});
server.listen(3000,()=>{
  console.log('server');
});
```

=> V8:

Use: module provides access to certain V8 engine features, such as heap statistics and memory usage.

- Properties & Methods:

V8.getHeapStatistics(): Returns an object with V8 heap statistics.

ex: var v8 = require('v8');
const hs = v8.getHeapStatistics();
console.log(hs);

=> OS:

- Use: module provides information about the computer's operating system.

- Properties & Methods:

arch(): returns the OS CPU architecture.
cpus(): Returns an array containing information about the computer's CPUs.
constants: returns an object for process signals, errors codes etc.
EOL: returns end-of-line marker.
platform(): returns OS platform.
type(): returns the name of the OS.
uptime(): returns uptime in seconds.
userInfo(): returns information about the current user.

ex:

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

=> zlib:

use: module provides a way of zip and unzip files.

. Properties & Methods:

constants: Returns an object containing zlib constants.
createDeflate(): creates a Deflate object.
createGzip(): creates a Gzip object.
deflate(): Compress a string or buffer.
unzip(): Decompress a string or buffer.

ex:

```
var zlib = require('zlib');
var fs = require('fs');
var gzip = zlib.createGzip();
var r = fs.createReadStream('./d1.txt');
var w = fs.createWriteStream('./d2.txt.gz');
r.pipe(gzip).pipe(w);
```