Page No.
Date

Theory Assignment : 01

Name : Belim Atiya Mohammed Shoeb

Roll No : 002

Sem : 7th (M.Sc(IT))

Subject : 701 - Application Development using Full-stack.

Page No.
Date

Q:1 Nodejs : Introduction, Features, execution architecture.

- Nodejs is an open-source, cross platform runtime environment for executing Javascripts code outside of a web browser. It was created by Ryan Dalhi. Nodejs is built on the V8 Javascript engine, which is the same engine that powes the Google Chrome web browser.

One of the key features that makes Nodejs popular is its even-driven, non-blocking I/O model, which allows it to handler a large number of concurrent connection efficiency. This makes it ideal for application that involve real-time interactions, such as chat application, online games and streaming services

→ Features of Node.js :

- (1). Asynchronous and Non-blocking:
Node.js uses an event-driven, non-blocking I/O model, which allows it to perform asynchronous operations efficiency. This means that it can handle multiple request simultaneously without waiting for a response before moving on to the next one.

- (2). Server-side Javascript :
Node.js allows developers to use Javascript on the server-side, which provides a unified programming language for both client-side

and server-side development. This reduces the need for context switching between different language for different parts for the Application.

- (3) NPM (Node Package Manager)
  NPM is the default package manager for Node.js and is one of the largest software registers in the world. it provides easy access to thousand of open-source libraries and packages that can be used to extend the Functionality of Node.js applications.
- (4) Fast and Efficient
  Node.js is build on the V8 engine, which compile Javascript directly into machine code, making it Fast and performant. Additionally, its non-blocking I/O model enables it to handle a large number of concurrent connection efficiently.
- (5) Scalable
  Node.js is designed to be highly scalable making it suitable for application that require handling a large number of concurrent connection, such as real-time application.
- (6) Large Community and EcoSystem.
  Node.js has a large and active community of developers, which means there are plenty of resource, libraries and tools available to support and enhance Node.js development.

→ Execution Architecture:

Node.js follows a single-threaded, event-driven architecture. Let's break down its key components.

01. Event Loop: The event loop is the core of Node.js execution model. it continuously checks the event queue for pending events and execution the associated callbacks. When a request or operation is completed, a corresponding event is added to the event queue.

02. Event Queue:

This event queue holds all the events and their associated callbacks. Asynchronous operations, such as file I/O, network request or timers are processed in the background and their callbacks are placed in the event queue when they are ready to be executed.

03. Callback Functions:

Callback Functions are user-defined functions that are executed once a certain event or asynchronous operation is completed. They are essential for handling the result of asynchronous tasks.

04. Libuv

Libuv is a multi-platform support library that provides the core functionality for Node.js event loop and handles asynchronous I/O operations. it abstracts the differences in

I/O mechanism on different platforms, allowing Node.js to be cross-platforms.

05. Worker Threads (optional)
While Node.js is a single-threaded, it provides an optional API called "Worker Threads" that allows developers to create and manages multiple thread.

Q:2 Note on modules with Example.

- In Programming, modules are a way of organizing code into separate, reusable units of functionality. They allow developers to break down a large application into smaller, managable pieces, making the code more maintainable, scalable and easier to understand. Each module typically focuses on a specific task or functionality and it can be used in other parts of the app. without having to rewrite the same code.

→ Creating a Module in Node.js :
To create a module in Node.js, you define a Javascript file with the functionality you want to export and make it available to use in other parts of your application.

For Example, let's create a simple module that calculates the area of Rectangle :

- Let's create 'Rectangle.js' module.

```
function calculateArea(width, height)
  {
     return height * width;
  }
module.exports = calculateArea;
```

Now, Let's create a file to use this module,

```
const calculateArea = require('./rectangle');

const width = 5;
const height = 10;
const area = calculateArea(width, height);
console.log(`Area = ${area}`);
```

- The Output of following Program will be

Area = 50;

By breaking down the Functionality into seperate modules, we promote code reusability and maintainability, making it easier to manage and extend the application as it grows. Node.js allows you to create and use module effectively allowing developers to build complex application while keeping the codebase organized and maintainable.

Page No.
Date

Q:3 Note on package with example.

- In Node.js, a package refers to a collection of reusable code and assets bundled together in a standardized format. Package can contain libraries, and other resources that enhance the functionality of a Node.js application. Node.js uses a package manager called NPM (Node Package Manager) to help developers easily install, manage and share packages.

NPM and package.json:

- NPM is a command-line tool that comes bundled with Node.js, allowing developers to interact with the NPM registery, which contains a vast collection of open-source packages. it provides a simple way to install package and manage project dependencies.

When you start a new Node.js project, you typically create a 'package.json' file at the root of your project. These file contain metdadata about your project and lists all the dependencies required for your application to run.

- Let's create a Package.json:

first navigate to your project directory

Page No.
Date

In the terminal and run the command:

- npm init

This command will prompt you to enter various details about your project such as the package name, version, description, entry point, etc.

To install packages in your Node.js Project you can use the 'npm install' command followed by package name. For Example, to install node-static,

- npm install node-static

To use these packages in your program you can import them by providing package name.

For Example:

npm install lodash

```
- const a = require('lodash');
  const num = [1, 2, 3, 4];
  const sum = a.sum(num);

  console.log('The sum is : ${sum}');
```

Output:

The sum is : 10

- Updating and managing Packages:
  over time, packages may receive updates to fix bugs or add new feature To update package to their lastest version, you can use the 'npm update' command:

```
npm update
```

You can manage specific version of packages by editing the 'dependencies' section in the 'package.json' file or using the 'npm install' command with the '@' symbol followed by the version number.

Q: 4 Use of Package.json and packa-lock.json in Node-js packages

- In Node.js package, both 'package.json' and 'package-lock.json' are essential files used to manage dependencies and ensure consistency across different installtion of the project.

(1) Package.json
The package.json file is the heart of a Node.js project. It contains metadata about the project, including its name, version description, author and more. When you create a new node.js project or

initiatize one with 'npm init', a 'package.json' file is generated in the root directory

→ Key components of package.json:

- name : The name of the package.
- version : The version number of the P.
- description : A brief description of the Package.
- dependencies : A list of packages that the project depends on to run correctly in production.
- devDependencies : A list of packages that are only needed during development and testing, such as testing frameworks or build tools.
- Scripts : Custom Scripts that can be executed using 'npm run <script-name>'. These scripts can be used for various task, like running tests starting the server, building the Project.

(2) Package-lock.json:

The 'package-lock.json' is a automatically generated file that keeps track of the specific version of all the packages installed in your project including their transitive dependencies. It ensure that everyone working on the project is using the same version

of the packages, eliminating the "works on my machine" problem.

When you install a new package or update an existing one using 'npm install', 'npm update', or 'npm-ci', the 'package-lock.json' file is automatically updated to reflect the exact versions of the packages installed in your project. It locks the version down to prevent unintesional upgeades when others work on the same project.

In Summary, the 'package.json' file contain metadata about the project and it's dependencies, while the 'package-lock.json' file keeps track of speed specific package versions to ensure consistency among different installtions. Togther, these files forms an integral part of Node.js projects, making package managment more relaiable and straighforward.

Page No.
Date

Q:5 npm introduction and commands with its use:

- NPM - node package Manager is the default package manager for Node.js, providing a vast collection of open-source packages and tools that developers can use to enhance their Node.js projects. It is a command-line tool that allows you to easily install, manage and share packages and their dependencies. NPM makes it simple to integrate third-party libraries, modules and other resources into your projects.

→ NPM Commands and Their Use:

- 'npm init':
This command initializes a new Node.js project and creates a 'package.json' file interactively. It will prompt you to enter information about your project such as the package name, version, description, author, entry Point, etc.

- 'npm install <package-name>'
This command installs a specific package and adds it as a dependency to your 'package.json' file.

Page No.
Date

Ex: 'npm install express'

- 'npm install':
This command installs all the packages listed in the 'dependency' section of your 'package.json' file.

- 'npm install <package-name> --save-dev':
This command installs a package and adds it to the 'devDependencies' section of your 'package.json' File. Dev depencies are only required during development and testing.

Ex: 'npm install jest --save-dev'.

- 'npm update <package-name>:
This command updates a specific package to its latest version and updates the version in the 'package.json' file.
Ex: 'npm update lodash'

- 'npm restart':
This command is used to restart a package

- 'npm version':
This command shows the current npm version installed on your computer'.

Page No.
Date

Q:6 Describe use and working of following Nodejs packages.

01. url :
The 'url' module provides utilities for URL resolution and parsing. The getters and setters implements the properties of URL object on the class prototype and the URL class is available on the global object.

Syntax:

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

- url module splits up a web address into readable parts.

02. Process, pm2 (external package)
process object is a global which is an instance of EventEmitters and can be accessed directly.
- it can also be explicitly access using module global.
- process is a module which provides interaction with the current nodeJS process.

→ PM2 is a NodeJS process manager that comes with a build-In load balancer.
- it helps facilities production deployment and enables you to keep running applications alive and indefenitly.

- npm i PM2
it is used to restart after crashing.

03. readline :
The readline module in Node.js is a build-in package that provides an easy way to read input from users via the command line. it allows developers to interact with users by reading lines of text input and providing response.

const readline = require ('readline');
The working of readline :
- Creating the Interface
- Asking Question and Getting User Input
- Closing the Interface

04. Fs
The fs - file system module is a build-in package in Node.js that provides an API for working with the file system. It allows developers to perform various file-related operations, such as reading, from & writing from files, creating and deleting files and directories and more.

- The fs module provides synchronous and asynchronous method to perform file

Page No.
Date

System operation.

05. The 'events' module is used to implement event-driven programming in Node.js applications. It allows developers to define custom events and associated event handlers, making it easy to build scalable and loosely coupled applications.

To work with the 'event' module, you need to create an instance of the 'EventEmitter' class, which provides method to emit events and register event handlers.

06. console.

The console module in Node.js is used to log messages and interact with the standard output and standard error streams. It provides method like 'console.log()', 'console.error()', etc to print different types of messages to the console.

07. The 'buffer' module in Node.js is a build-in package that provides a way to handle binary data. It Allows developers to work with raw binary data in the form

of buffers, which are fixed size chunks of memory that can store binary data.

The primary Usage of the 'buffer' module is to work with the binary data. Buffers are especially useful when dealing with streams and when converting between data format.

08. Querystring

The querystring module in Node.js is a build-in package that provides utilities for parsing and formatting URL query string.

The Usage of querystring module is to work with URL query. It is commonly used in web application to parse the query parameter from URL.

- querystring.Parse(str[, sep[, eq[, option]]]);

This method is used to parse a query string and convert it into a JavaScript Object.

09. The http module in Node.js is a build in package that allows developers to create HTTP servers, make http request and interact with the http protocal.

The primary use of http module is to build http server and clients. It

provides classes and methods to handle incoming http requests on the server side and sent http request on client side.

10. V8

The V8 module in Node.js provides access to V8 engine-specific functionality. It allows developers to gather memory and performance-related data and control certain aspects of V8's behaviour.

This module is mainly is used for advanced memory managment.

11. os.

This module is used to provide information about the computer operating system. it provides functions to interact with the operating system.

It provides the hostname of os and returns the amount of free system memory in bytes.

12. zlib

It is used to in Node.js to compress or decompress a file. Compress means zip the file & Decompress means to unzip the file.

Date

It is a pre-build Node.js module so not required to install it as a dependency.

07 Important Propeerties and methods and relevant Progeam.

01. url :
- Propeerties : 'url.parse()', 'url.format()'
- Method : 'url.resolve()'

Program :
```
const url = require ('url');
const urlstring = "https://www.example.com
                   /path?name=John';
const parseUrl = url.parse(urlstring, true);
console.log(url.format(parsedUrl));
console.log(url.resolve('/api', 'users'));
```

02. process, pm2 (external package)
- properties : 'process.argv', 'process.env'
- Method : 'process.exit()', 'pm2.start()

Program :
```
console.log(process.argv);
console.log(process.env.NODE_ENV);

const pm2 = require('pm2');
pm2.start(
```

```
script : 'app.js' },
  (err, apps) => {

if(err) throw(err);
 pm2.disconnect();
});
```

03. readline :

- properties : N/A
- Methods : 'readline.createInterface()',
  'rl.question()'.

Program :

```
const readline = require('readline');
const url = readline.createInterface({
  input : process.stdin
  output : process.stdout
  });
url.question('name ?', (name) => {
  console.log('${name}');
   url.close();
});
```

04. fs :

- Methods : 'fs.readFile()', 'Fs.writefile()',
  'fs.readdir()'.

Page No.
Date

Program :

```
const fs = require ('fs');
fs.readfile ('File1.txt', 'utf8', (err, data)
        => { if (err throw err;
  console.log (data);
});
fs.writefile ('file1.txt', 'Hello' (err) => {
  if (err) throw err;
  console.log ("success");
});
```

05. Events :

Methods : addListener(), on(), once(), emit()

Program :

```
const EventEmitter = require ('events');
var event = new EventEmitter ();
event.on ('MyEvent', (Msg) => {
    console.log (Msg); });
```

06. Console.

Method : 'console.log()' 'console.error()',
'console.warn()'

Program :

```
console.log ("This is Message");
console.error ("error"),
console.warn ("Warning");
```

07. Buffers

Methods : 'Buffer.from()', 'Buffer.concate()'

Program :

```
const arrbuf = new ArrayBuffer(16);
const buffer = Buffer.from(arrbuf);
if (buffer.buffer === arrbuf)
    console.log("both are equal");
else
    console.log("not equal");
```

08 Querystring

Methods : 'querystring().parse()', 'querystring.stringify()';

Program :

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

09. http

Method : 'http.createServer()', 'http.request()'

Program :

```
var http = require('http');
http.createServer(function(req, res) {
    res.write('Hello');
    res.end();
}).listen(8080);
```

10. V8

Method : 'v8.getHeapStatistics()', 'v8.setFlagsFromString()'

Program :

```
const v8 = require('v8');
console.log(v8.getHeapStatistics());
v8.setFlagsFromString('--max-old-space-size=8192');
```

11. OS

Properties : 'os.hostname()', 'os.platform()'

Method : 'os.cpus()', 'os.totalmem()'

Program :

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

12. Zlib :

Method : 'zlib.createZip()', 'zlib.createGunzip()'

Program :

```
var zlib = require('zlib');
var fs = require('fs');
var gzip = zlib.createGzip();
var r = fs.createReadStream('./demo.txt');
r.pipe(gzip).pipe();
```