Theory Assignment - 1

Name : Nishi Y. Sharma

Roll no. : 72

Semester : 7th

Subject : Application Development using Full Stack (701).

* Nodejs : Introduction, Features, execution architecture.

- Introduction :

→ Node.js is a cross platform runtime environment and library for running Javascript applications outside the browser. It is used for creating server-side and network web applications. It is open source and free to use.

→ Many of the basic modules of Node.js are written in javascript. Node.js is mostly used to run real-time server application.

→ Node.js also provides a rich library of various Javascript modules to simplify the development of web applications.

Node.js = Runtime Environment + Javascript library

* Features of Node.js

1. Extremely fast :
Node.js is built on Google Chrome's V8 Javascript Engine. so its library is very fast in code execution.

2. Single threaded :
Node.js follows a single threaded model with event looping

3. Highly Scalable :
Node.js is highly scalable because event mechanism helps the servers to respond in a non-blocking way.

4. No Buffering ;

Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.

5. Open Source :
Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.

6. License :
Node.js is released under the MIT license

7. I/O is Asynchronous and Event Driven :
All APIs of Node.js library are asynchronous, i.e. non-blocking.

- Execution Architecture

→ Node.js follows a unique execution architecture that enables its non-blocking, asynchronous behavior.

1. Event loop :
The event loop is responsible for handling asynchronous operations and I/O tasks efficiently. It keeps an eye on the message queue and executes the callback functions associated with completed I/O operations when the call stack is empty.

2. Single - Threaded, Non-Blocking :
Node.js operates on a single-threaded event loop. This might seem counterintuitive for handling multiple concurrent requests, but Node.js achieves this through non-blocking I/O operations.

3. Callback and Asynchronous programming :
Callbacks play a significant role in Node.js. Instead of waiting for a function to complete before moving on to the next one, Node.js uses callbacks to notify the completion of an asynchronous task.

4. Event-Driven Architecture.
Node.js utilize an event-driven architecture to handle events and execute associated callback when an event occurs. Events can be anything from I/O operations to HTTP requests.

5. libuv :
Node.js's relies on a library called libuv to handle its asynchronous I/O operations. libuv ~~est~~ abstracts the underlying system calls and provides a consistent interface across different platform. It manages threads, handles file I/O, networking, and timers efficiently to optimize the event loop's performance

Q-2

* Note on modules with example.

→ In Node.js, modules are reusable blocks of code that encapsulate functionality and can be imported and used in other parts of the application. Modules help in organizing code, promoting code reusability and maintaining a clean codebase. There are different types of modules in Node.js, including built-in core modules, local modules, and third-party modules.

1. Built-in Core Modules :
Node.js comes with several core modules that provide essential functionalities. These modules can be used directly without need for any additional installation.

Ex.

```
const fs = require('fs');
fs.readFile('example.txt', 'utf8', (err, data) =>
    {
        if(err) {
            console.error('Error reading file', err);
        } else {
            console.log('File contents', data);
        }
    });
```

→ This is the example using the 'fs' (file system) core module to read file.

2. Local Modules :
Local Modules are custom modules created by developers within their projects. These modules ~~are~~ can be imported into other files using relative paths. Let's create simple local module.

'rectangle.js'

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

3. Third-party Modules (npm modules) :
Node.js has a vast ecosystem of third-party modules available on the npm registry. Developers can install these modules and use them in their projects. Let's use 'axios' mo

```
npm install axios.
```

```
const axios = require ('axios');
axios.get ('http://api.example.com/data')
  .then ((response) => {
    console.log ('Data received', response.data);
  })
  .catch ((error) => {
    console.log ('Error fetching data', error.message);
});;
```

Q-3

* Note on package with example.

→ A package in Node.js contains all the files you need for module.

→ A package is a folder tree described by a package.json file. The package consists of the folder containing the package.json file and all subfolders until the next folder containing another package.json file, or a folder named node_modules.

→ This page provides guidance for package authors writing package.json file along with a reference for the package.json.

→ NPM creates a folder named "node_modules" where the package will be placed. All packages you install in the future will be placed in this folder

C:\Users\My Name\node_modules\upper-case

Q-4

* Use of package.json and package-lock.json

→ In Node.js, package.json is a versioning file used to install multiple packages in your project. As you initialize your node application, you will see three files installed in our app node_modules, package.json and package.lock.json.

package.json	package.lock.json
→ It contains basic information about the project.	It describes the exact tree that was generated to allow subsequent installs to have identical tree.
→ It is mandatory for every project.	Operations where npm modifies either node_modules tree or package.json.
→ It records important metadata about the project.	It allows future devs to install the same dependencies in the project.
→ It contains information such as name, author, description, script and dependencies.	It contains the name, dependencies and locked version of the project.

Q-95

* Nodejs packages :-

→ In the Nodejs ecosystem, packages are collections of modules, libraries, and resources that developers can use to enhance their applications.

→ Node.js packages provide functionalities for various purposes, ranging from web development and server-side tasks to command-line utilities and more.

→ Web Frameworks :
Packages like: Express.js, koa and Hapi are popular web frameworks that simplyfy the process of building web applications and APIs by providing routing, middleware support and other features.

→ Utility libraries :
Packages like lodash, Ramda and Underscore.js provide utility functions that assist with tasks like data manipulation, validation and functional programming.

→ Database Libraries :
Packages like Mongoosejs and Sequelize provide easy-to-use abstractions for working with database and ORM (Object Relation. Mapping) capabilities.

→ Authentication and Security :
Packages like Passport.js and bcrypt offer solutions for authentications and password hashing to enhance application security.

→ Template Engines :
Packages like EJS, Handlebars and Pug enable developers to generate dynamic HTML content easily.

→ HTTP Clients :
Packages like Axios and Request provide tools for making HTTP requests, allowing Node.js applications to interact with APIs and web services.

→ Testing Frameworks :
Packages like Mocha, Jest and Chai offer testing utilities and assertions for creating and running test suites.

→ CLI (Command-Line Interface) Tools :
Packages like commander and yargs enable developers to build interactive and user-friendly command-line tools.

Q-6 NPM introduction and commands with its use

→ NPM is the default package manager for Node.js and it is one of the largest software registers in the world. It allows developers to easily install, manage, and distribute Node.js packages to be used in their projects, npm comes bundled with Node.js, so when you install Node.js, npm is automatically installed on your system.

→ npm init :
This command initializes a new Node.js project and creates a 'package.json' file. It prompts you to enter details about the project such as name, version, description, author and entry point.

→ npm install <package-name>
Installs a specific package and adds it to the 'dependencies' section in 'package.json'.

→ npm uninstall <packagename>
Removes a package from the project and update the 'packagejson' file accordingly.

→ npm update :-
Updates all the packages listed in 'package.json' to their latest versions based on the specified version ranges.

→ npm search <package-name>
Searches the npm registory for packages with the given name

→ npm ls :-
Lists all the installed packages in the current project.

→ npm init -y :-
Initializes a new range project with default values, skipping the prompts. It creates a 'package.json' file with default settings.

→ npm publish :
Publishes a package to the npm registory making it available for others to use.

Q.7 Describe use and working of following Node.js packages, Important properties and methods and relevant programs.

1) url :
The 'url' module provides utilities for URL resolution and parsing. It is used to work with URLs and extract information from them.

Eg Parsing a URL :

```
const {URL} = require('url');
const urlstring = 'https://www.demo.com:8080/path?query=http#';

const parsedUrl = new URL(urlString);
Console.log(ParsedURL.host);
console.log(parsedURL.pathname);
console.log(parsedUrl.searchParams.get('query'));
```

2) process pm2 (external package):
'process' object provides info. & control every the Node.js process. It allows interacting with the current process and accessing environment variables

Getting Command.line Arguments.

```
Console.log(process.argv);
```

- 'pm2' is an external package used to manage Node.js processes, It provides tools for process monitoring, sealing and cluster management.

```
# Install pm2 globally
npm install -g pm2
pm2 start app.js.
```

3). readline :-

The 'readline' module provides an interface for reading input streams line by line. It is commonly used to interact with users in the command.line environment.

```
const readline = require ('readline');
const rl = readline.createInterface ({
    input : process.stdin;
    output : process.stdout
});
rl.question ('What's your name ?', cname) =>
  {
    Console.log ('Hello, ${name} !');
    rl.close ();
});
```

4) 'fs'

The 'fs' module provides file system related functionality, allowing reading, writing, and manipulating files.

Reading a File :-

```
const fs = require('fs');
fs.readFile('example.txt', 'utf-8', (err, data) =>
{
    if(err) {
        Console.error('Error reading file:', err)
        return;
    }
    console.log('File content', data);
});
```

5). events :

The 'events' module provides an event-driven architecture for building applications that can emit and listen to events.

→

```
const EventEmmiter = require('events');
Class MyEmitter extends Event Emitter {}

const myEmitter = new myEmitter();
myEmitter.on('greet', (name) => {
    Console.log('Hello, ${name}!');
});
myEmitter.emit('greet', 'Nishi');
```

6). Console :
The 'console' module provides a simple debugging console that can be used to log messages during development.

```
Console.log ('This is a log message');
Console.error ('This is an error message');
Console.warn ('This is a warning message');
```

7). buffer :
The 'buffer' module provides a way to handle binary data. It is used to work with raw binary data in Node.js applications.

```
var buf = Buffer.from ('abc');
Console.log (buf);
```

Ans : < Buffer 61 62 63 >

8) querystring :
The 'querystring' module provides utilities for working with query strings in URLs.

```
Const querystring = require ('querystring');

const params = querystring.parse ('name=Nishi
                                   & age=20');
console.log (params);
```

Output : { name : 'Nishi', age : '20' }

9). http :
The 'http' module provides a set of functions and classes to create HTTP servers and make HTTP requests.

```
const http = require ('http');
const server = http.createServer ((req, res)=>
{   res.writeHead (200, {'Content-Type': 'text/plain'});
    res.end ('Hello, World!');
});
server.listen (300, () => {
    Console.log ('Server is running on port
                                    300');
});
```

10). V8 :
The 'V8' module exposes APIs, related to the V8 javascript engine, providing acess to performance and memory-related data.

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

11). OS :
The 'OS' module provides operating system related functionality, such as information about the host operating system.

```
const os = require ('os');
console.log ('os platform:', os.platform())
Console.log ('CPU Architecture:', os.arch());
```

12). zlib :-

The 'zlib' module provides compression and decompression functionalities using gzip and deflate.

Compressing and Decompressing :-

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

const data = 'This is same data to compress';
zlib.gzip(data, (err, compressedData) =>
{ if(err) {
      console.error('compression error', err);
      return;
  }

  console.log('Compressed data:', compressedData);

  zlib.unzip(compressedData, (err, decompressed
                                    Data) =>
  {
    if(err) {
       console.error('Decompression error', err);
       return;
    }
    console.log('Decompressed Data:'
                  decompressedData.toString());
  });
});
```