Theory Assignment - 1

Name : Parmar Kirti R.

Roll No : 39

Semester : M.Sc.IT [7th sem]

Subject : Application Development using full stack

Que.1 Nodejs : Introduction, features, execution architecture

Ans : • Introduction :

- Node.js is an open-source, cross-platform Javascript runtime environment that allows developers to execute Javascript code outside of a web browser. It was created by Ryan Dahl in 2009 and has since become a popular co choice for building scalable, high-performance and real-time application. Node.js is built on Chrome's V8 Javascript engine, which provides a fast and efficient runtime for executing Javascript code.

• Feautures :

1) Asynchronous and Event-Driven :

Node.js is designed to handle a large number of concurrent connections efficiently. It follows an event-driven, non-blocking I/O model, allowing applications to perform multiple tasks concurrently without waiting for each operation to complete, which can significantly improve performance.

2) Package Ecosystem :

Node.js has a vast and vibrant ecosystem of open-source packages available through the Node Package Manager (NPM). NPM allows developers to easily access and integrate third-party libraries and modules into their projects, speeding up development and encouraging code reuse.

3) Single-Language Development :

Node.js enables developers to use Javascript both on the server-side and the client-side. This language consistency reduces the need for context switching and makes it easier for full-stack developers to work on both ends of an application.

4) Lightweight and scalable :

Due to its non-blocking architecture, Node.js can handle a large number of concurrent connections with relatively low resource usage, making it highly scalable and suitable for building real time applications and microservices.

5) Cross-Platform :

Node.js is compatible with various operating systems, including windocos, macOs and various Linux distributions, making it easy to deploy and run applications on different platforms.

- Execution Architecture :

- The execution architecture of Node.js is based on its event-driven, non-blocking I/O model. Here's the key components and How they work;

1) Event Loop :

At the core of Node.js is the event loop. It is responsible for handling all I/O operations and managing asynchronous events. The event loop continuously listens for events, such as incoming requests, file read/write operations, or timers, and delegates the appropriate callbacks to the corresponding event handlers.

2) Callbacks :

callbacks are functions passed as arguments to other functions. They are a fundamental concept in node.js, allowing developers to define actions that should be taken once a certain operation is completed. For example, when reading a file, you can provide a callback function to be executed once the file has been read.

3) Non-Blocking I/O :

Node.js utilizes non-blocking I/O operations, meaning it does not wait for an I/O operation to complete before moving on to the next task, instead, it continues to execute other tasks and registers a callback to be called when the I/O operation is finished.

4) Event Handlers:

Event handlers are functions that respond to specific events, such as HTTP requests, timers, or file system events.

5) Event Queue:

Event queue holds all the callbacks associated with completed I/O operations and event. When an asynchronous operation completes, its callback is placed in the event queue, waiting to be processed by the Event Loop.

6) Microtask queue:

Node.js also has a microtask queue that holds microtasks - tasks with higher priority than regular callbacks. Microtasks are usually related to promises and specific APIs like 'process.nextTick()'. The microtask queue is processed before the event queue, ensuring that certain callbacks are handled with higher priority.

Que.2 Note on modules with example.

Ans. - In Node.js, modules are a fundamental concept used to organize and encapsulate code into reusable components. Each file in Node.js is treated as a module and the code within a module is kept private by default. To make elements from one module accessible to another module, you need to explicity export them from the source module and import them in the destination module.

- Example: 'math.js'

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

function subtract (a, b) {
    return a - b;
}

exports.multiply = function (a, b) {
    return a * b;
};

exports.divide = function (a, b) {
  if (b === 0) {
    throw new Error ("can't divide by zero");
  }
  return a / b;
};
```

Que.2. Note on package with example.

Ans - In Node.js, a package refers to a collection of code, usually in the form of modules, that is organized and distributed together. Packages can include various functionalities, utilies, and libraries and they are managed using the Node Package Manager (NPM).

- NPM is a powerful tool that allows developers to install, manage and share packages easily. It comes bundled with Node.js, so there's no need to install it separately. NPM maintains a vast repository of public packages, and it also allows developers to publish their own packages for others to use.

- Example Package : 'lodash'

- 'lodash' is a popular utility library that provides a wide range of useful functions to work with arrays, objects, strings and more.

⇒ Installing Package:

- To use a package in your Node.js project, you need to install it first. You can do this using the NPM command-line tool in project's directory.

- npm install lodash

- This will download the latest version of the 'lodash' package and add it to the 'node_modules' directory of your project. It will also create or update the 'package.json' file to include the 'lodash' package as a dependency.

⟹ Using the package:

- Once the package is installed, you can use it in your node.js code:

```
const _ = require('lodash');

const numbers = [1, 2, 3, 4, 5];
const sum = _.sum(numbers);

const person = {
    name: 'John',
    age: 30,
    city: 'New York'
};

const keys = _.keys(person);
```

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

Ans. • package.json :

- The 'package.json' file is a JSON file that contains metadata about your Node.js project and its dependencies. It servers as a manifest for the project and includes information such as the project's name, version, author, license, scripts, and most importantly the list of dependencies required to run the project.
- The 'package.json' file is typically located at the root of your project directory. It can be created manually or generated using the npm init command.

• package-lock.json :

- The 'package-lock.json file is automatically generated by npm when you install or modify dependencies in your project. It is designed to provide a detailed and deterministic record of the exact versions of packages that are installed in the 'node_modules' directory.
- This file ensures that all developers working on the project use the same version of dependencies, preventing potential conflicts or unexpected behavior due to version mismatches.

Que 5. Node.js packages :

1) Web Frameworks :
- Express : A minimalist and flexible web framework for building web applications & APIs.
- Koa : A modern and lightweight web framework designed by the creators of Express.
- Hapi : A powerful framework for building applications

2) Utility Libraries :
- Lodash : A utility library that provides helpful function for working with arrays, objects, strings & more
- Moment.js : A library for parsing, validating, manipulating and formatting dates & times

3) Database Integration :
- Mongoose : A popular ODM library for MongoDB, making it easier to work with Database
- Sequelize : An ORM library for SQL databases like PostgreSQL, MySQL & SQLite

4) Authentication & security :
- Passport : A flexible authentication middleware for Node.js supporting various strategies (local, ...
- Bcrypt : A library for hashing & salting passwords to securely store them in databases

5) Real - Time & Websockets :
- socket.io : A library for enabling real-time communication between clients and servers using websockets.

6) Testing :
- Mocha : A feature-rich testing framework for writing test suites and conducting test cases
- Chai : An assertion library that integrates well with mocha to make writing test assertions easier.

7) Logging :
- Winston : A versatile logging library with support for logging to different transports.
- Bunyan : A fast and simple JSON logging library.

8) HTTP Clients :
- Axios : A popular and easy-to-use HTTP client for making API requests.
- Request : A simple HTTP request library for Node.js

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

Ans. • NPM Introduction :

NPM stands for Node Package Manager and it is the package manager for the Node JavaScript platform. It put modules in place so that node can find them and manages dependency conflicts intelligently. Most commonly, it is used to publish, discover, install and develop node programs.

• npm commands :

- npm install : Installs a package in the package.json file in the local node_modules Folder.

- npm uninstall : Remove a package from the package.json file and removes the module.
- npm update : Update the specified package. If no package is specified then it updates all the package in the specified location
- npm update -g : Global update command will apply the update action to each globally installed package.
- npm outdated : checks the registry if any package is outdated. It prints a list of all package which are outdated.
- npm doctor : checks our environment so that our npm installation has what it needs to manage our Javascript packages.
- npm init : Initialize command creates a package.json file in our directory.
- npm start : Runs a command that is defined in the start property in the scripts.
- npm build : It is used to build a package.
- npm ls : List command lists all the packages as well as their dependencies installed.
- npm version : Bumps a package version.
- npm search : searches the npm registry for packages matching the search terms.
- npm help : searches npm help documentation for a specified topic. It is used whenever the user needs help to get some reference.
- npm owner : manages ownership of published packages. It is used to manage package owners.

Que.7. Describe use, important properties, methods, relevant programs and working of following Node.js Packages.

url, process, pm2 (external package), readline, fs, events, console, buffer, querystring, http, v8, os, zlib.

Ans. 1) url :

- Use of the 'url' module :

- The primary use of the 'url' module is to parse URLs and access their individual components, such as protocol, hostname, path, query parameters and fragments. It provides methods to format and resolve URLs.

- Important properties and methods of the 'url':

- 'url.parse(urlString[, parseQueryString[, slashesDenoteHost]])':

This method is used to parse a URL string and returns an object containing the various components of the URL. The 'parseQueryString' and 'slashesDenoteHost' parameters are optional and control how the query string and leading slashes are handled.

- 'url.format(urlObject)':
  This method takes a URL object and converts it back into a URL string.

- 'url.resolve(from, to)':
  This method resolves a relative URL 'to' based on the abo obsolute URL 'from'.

- Relevant Programs and Working:

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

const urlObject = {
    protocol : 'https:',
    hostname : 'www.example.com',
    port : '8080',
    pathname : '/path/to/resource',
    query : { id : '123' },
    hash : '#section'
};

const formattedUrl = url.format(urlObject);
console.log(formattedUrl);
```

2) process, pm2 (external package):

- Use of the process:

- The process module is used to interact with the Node.js process, get information about the runtime environment & handle events & signals.

- Important properties and methods:

- process.argv:
  An array that contains command-line arguments passed to the Node.js process. The first two elements of this array are the path to Node.js and path to the script being executed, and the subsequent elements are the command line arguments.

Que.

- process.env:
  An object representing the environment variables of the current process.

Ans

- process.cwd():
  Returns the current working directory of the Node.js process.

- Relevent program and working:

- console.log(process.env.NODE_ENV);

- Use of pm2 (Process Manager 2):

- PM2 is used to manage and monitor Node.js applications in production to ensure they stay online, stable & efficiently use system resources.

- Relavent Programs and working:

npm install pm2 -g : install PM2 globally

pm2 start app.js : start a node.js application with pm2

pm2 list ; list running processes

pm2 logs ; monitor logs

pm2 reload app.js ; Reload application

pm2 restart app.js ; Restart application

pm2 stop app.js ; stop an application

3) readline :

- Use of the readline :

- The primary use of the readline module is to read input from users or other readable streams and process it line by line.

- Important properties and methods :

- readline.createInterface :

This method creates a new 'readline.Interface instance. The 'options' objects allows you to customize the input and output streams, prompt and other settings.

- Readline.Interface methods :

- resume() : Resumes the input stream after it has been paused.
- close() : closes the readline.Interface, releasing any associated resources.

• Relevent Program and working :

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

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

5) fs :

• Use of fs :

- The primary use of the 'fs' module is to interact with the file system to read, write, manipulate files and directories.

• Important properties and methods :

- fs.readFile(path[, options], callback):
  Asynchronously reads the contents of a file. The callback function is called with two arguments: 'err' and 'data'.

- fs.writeFileSync(path, data[, options]):
  Synchronously writes data to a file. It returns undefined if the write operation is successful or throws an error if it fails.

- fs.unlink(path, callback):
  Asynchronously remove a file.

- fs.stat(path, callback):
  Asynchronously gets the file status of file.

• Relevant program and working:

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

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

6) console:

• Use of console:

- The primary use of 'console' is for printing messages and logging various types of data to the console.

• Important Properties and Methods:

- console.log([data][,...args]);
  Prints the data to the console, followed by a newline. This is common method for logging.

- console.error([data],...args]:
  similar to console.log(), but the output is printed to 'stderr', which is typically used for printing error messages.

- Relevant Program and working:

```
console.log('Hello, World!');
console.error('This is an error');
```

7) buffer:

- Use of buffer:

- The primary use of buffer class is to ~~bonds~~ handle binary data, including encoding, decoding and manipulation of raw data in the form of bytes.

- Important properties and methods:

- Buffer.alloc(size[, fill[, encoding]]):
  Creates a new buffer of the specified size, filled with zeros. You can optionally provide a 'fill' value or 'encoding'.

- buffer.length:
  Returns the length of the buffer in bytes.

- Relevant program and working:

```
const buf1 = Buffer.alloc(10);
const buf2 = Buffer.from('Hello', 'utf8');
const buf3 = Buffer.from([0x48, 0x65, 0x6c,
                          0x6c, 0x6f]);
```

8) querystring:

- Use of querystring:

- The use of querystring is to handle URL query strings, parse them into Javascript objects & stringify Javascript objects into query string.

- Important properties and methods:

- querystring.escape(str):
  Escapes a string, encoding special characters and making it safe for inclusion in a URL query string.

- querystring.unescape(str);
  Unescapes a string, decoding any encoded characters in a URL query string.

- Relevant Program and working:

```
const querystring = require('querystring');
const querystring = 'name=abc&age=30';
const parsedObj = querystring.parse(querystring);
console.log(parsedObj);
```

10) V8 :

- Use of V8 :

- The use of V8 to execute Javascript code.
- The V8 package is not part of the standard Node.js modules and is not exposed directly to the users.
- The V8 Javascript engine is the open-source Javascript engine developed by google.
- while you cannot directly use the V8 package in your Node.js programs. it is crusial to understand its significance.

- Important ~~feaut~~ features and methods:

- Just-in-Time compilation:
- Garbage collection
- ECMAScript support
- Asynchronous Execution

11) OS :

- Use of OS :

- The primary of OS is to retrieve information about the operating system such as platform, architecture, network interface and system memory.

- Important properties and methods:

- os.platform():
  Returns the os platform (e.g., drowin, linux)

- os.arch():
  Returns the CPU architecture of the os.

- os.cpus():
  Returns an array of objects containing information about each CPU/core available on the system.

- os.totalmem():
  Returns the total amount of system memory in bytes.

- Relevant programs and working:

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

console.log('Platform:', os.platform());
console.log('Architecture:', os.arch());
```

12) zlib:

- Use of zlib:

- The use of zlib is to compress and decompress data using various compression algorithm, such as Gzip, Deflate and Brotli.

PAGE NO.
DATE : / /

- Important properties and methods :

- zlib.deflate(buffer, callback) :
  compresses the buffer using the Deflate algorithm and calls the callback with the compressed data.

- zlib.deflateSync(buffer) :
  synchronously compress the buffer using the deflate algorithm and returns the compressed data.

- zlib.deFlateRaw(buffer, callback) :
  compresses the 'buffer' using the raw Deflate algorithm and calls the callback with the compressed data.

- Relevant program and working :

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

const inputString = 'Hello!';
const inputBuffer = Buffer.from(inputString);

zlib.gzip(inputBuffer, (err, compressedData) => {
  if (err) {
    console.error('compression error:', err);
  } else {
    console.log('compressed Data:',
    compressedData.toString('baseG4'));
  }
});
```