VALENTINE
Page No.:
Date: / /

Name :- Maitri Bhajiyawala

Roll No :- 003

Semester :- II-7th

Subject :- Application Development using Full Stack - 701

Assignment :- 1

VALENTINE
Page No.:
Date: / /

1:1 Node js : Introduction, Features, Execution, architecture.

Introduction

→ Node.js is a server-side platform built on Google chrome's JavaScript Engine.
→ Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36.
→ Node.js is a platform built on chrome's JavaScript runtime for easily building fast and scalable network application. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time application that run accross distributed devices.
→ Node.js is an open source, cross-platform runtime environment for developing server side and networking applications.
→ Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS.
→ Node.js also provide a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

VALENTINE
Page No.:
Date: / /

Features

→ Asynchronus and Event Driven:
- All APIs of Node.js library are asynchron that is non-blocking. It essentially means a Node.js based server never waits for a API to return data.
- The server moves to next API after callin it and a notification mechanism of events Node.js helps the server to get a response f the previous API call.

→ Very Fast:
- Being built on Google Chrome's JavaScrip Engine, Node.js library is very fast in code execution.

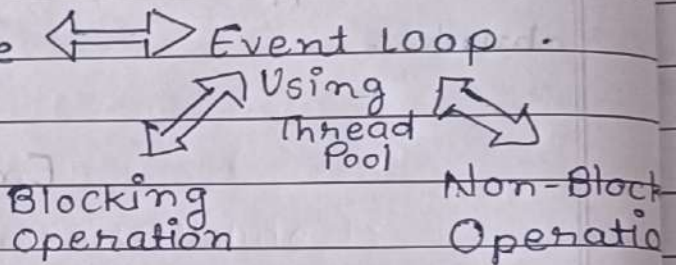
→ Single Threaded but Highly Scalable:
- Node.js uses a single threaded model with event looping. Event mechanism helps the serve to respond in a non-blocking way and ma the server highly scalable as oppesed to traditional servers which create limited thr to handle requests.
- Node.js uses single threaded program and t same program can provide services to a much larger number of requests than tradi nal servers like Apache HTTP server.

→ No Buffering:
- Node.js application never buffer any data. These application simply output the data in chunks.

Execution Architecture

→ Components of the Node.js Architecture:
- Requests: Depending on the actions that a user needs to perform, the requests to the server can be either blocking or non-blocking.
- Node.js Server: The Node.js server accepts user requests, processes them, and returns results to the users.
- Event Queue: The main use of Event Queue is to store the incoming client requests and pass them sequentially to the Event loop.
- Thread Pool: The thread pool is a Node.js server contains the threads that are available for performing operations required to process requests.
- Event Loop: Event loop receieves requests from the Event Queue and sends out the responses to the clients.
- External Resources: In order to handle blocking client requests, external resources are used. They can be of any type.

→ Workflow of Nodejs Server :



- Users send requests to the server for performing operations.
- The requests enter the Event Queue first at the server-side.
- The Event queue passes the requests sequentially to the event loop. The event loop checks the nature of the request.
- Event loop processes the non-blocking reque which do not require external resources and returns the responses to the correspond clients.
- For blocking requests, a single thread is assig to the process for completing the task by using external resources.
- After the completion of the operation, the req is redirected to the Event Loop which delivers the response back to the client.

Note on modules with example.

A set of functions you want to include in your application.

There are two types of modules:

① Built-in Module ② User-define Module.

Built-in Module:

Node.js has a set of builtin modules which you can use without any further installation.

You can include module using the require() function.

Syntax: require('Module Name');

Ex: var http = require('http');

```
http.createServer(function(req,res){
    res.writeHead(200,{'Content-Type':'text/html
    res.end('Hello');
}).listen(8080);
```

User-define Modules:

You can create your own module and use in your program.

You can use it with 'export' keyword to r properties and methods available outside module file.

File Name : FirstModule.js

Ex: exports.myDateTime = () => {
    return Date();
};

- Using Module.

```
var dt = require('./FirstModule');
var http = require('http');

http.createServer((req, res) => {
    res.write("Date & Time :" + dt.myDate
    res.end();
}).listen(8080);
```

Q:3 Note on package with example.

→ A package refers to a collection of modules functionalities bundled together for easily distribution and reuse.

→ Node.js uses npm (Node package Manage to manage packages.

→ npm is a command-line tool that comes bund with Node.js and allows you to install, publ and manage packages.

→ To create a new package
- npm init
- It will prompt you to give some informat about the package.

→ You can install packages which are predefined.
- npm install package-name
→ You can use installed packages.

Examples :-

```
var http = require('http');
var uc = require('upper-case');
http.createServer((req, res) =>
{
    res.write(uc.upperCase("Hello"));
    res.end();
}).listen(8080);
```

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

* Use of package.json :-

1. Managing Dependencies :
- To keep track of your project's dependencies, which are external modules or packages your project relies on.
- These dependencies can be other node.js packages that provides various functionalities, libraries or frameworks.
- When you define dependencies, it allows you and others to easily install all the packages by running a single command.

2. Defining Scripts:
- Allows you to define custom scripts that can b executed using the Node.js command.
- This scripts can be used to automate common tasks, such as running your application; runni tests, building the project.

3. Project Metadata:
- It containts metadata about your project, such its name, version, description, author, license and more.

4. Project Initialization:
- 'npm init' command guides you through a serie question to create the initial package.json fil

*. Use of package-lock.json:-

1. Dependency version tracking:
- The package-lock.json file keeps track of the exa versions of dependencies and sub-dependencies th are currently installed in Node.js project.

2. Constraint Builds:
- The File records the specific versons of depen it ensures that everyone working on the proje has the same dependencies installed, which helps t avoid version conflicts and ensures that the proj can be built and run consistently across different environments.

3. Faster and more reliable installs:
- The file allows npm to quickly and accurately install the same versions of dependencies accross different machines.

4. Security:
- The file help to prevent malicious code injection by ensuring that only verified and secure versions of dependencies are installed.

Q:5 Node js packages:

→ A package refers to a collection of modules or functionalities bundled together for easily distribution and reuse.

→ Nodejs uses npm (Node Package Manager) to manage packages.

→ npm is a command-line tool that comes bundled with Nodejs and allows you to install, publish and manage packages.

Q:6 npm introduction and commands with its u

→ npm is a short form of Node Package Mana

→ It is acts as a command line utility for t the Node.js project for installing packages in the project, & dependency management, an even version management.

- npm : To locate the npm Api on your computer.
- npm -v : To check the version of the npm.
- npm init : To create a new package.
- npm install <package_name> : To install a new npm package.
- npm i <package-name> : To install a new package.
- npm install <package-name> --save-dev : To install packages as a requirement for development
- npm install <package_name> -g : To install packa globally.

Q:7 Describle working and use of following Nodejs packages.

- Important properties and methods and relevant programs.

1). url :-

→ Node.js has a built-in "url" module that provides utilities for URL Resolution and parsing.

→ The 'url' module commonly used in Node.js application to work with URLs parse them into their respective components.

→ Methods :-

* url.parse(urlString[, parseQueryString[, slashesDenotHost]]):

- This method is used to parse a URL string and return an object containing its various components.

* url.format(urlObject):

- This method takes a URL object returned by url.parse() and converts it back into a URL string.

* url.resolve(from, to):

- This method resolves a relative URL 'to' to an absolute URL using the 'from' URL as the base.

→ Properties:

* protocol: The URL protocol.
* slashes: A Boolean indicating whether the pr
  has double slashes.
* auth: The authentication information.
* host: The full host portion of the URL.
* hostname: The hostname of the URL.
* pathname: The path of the URL.
* search: The query string portion of the URL,
  including the leading '?'.
* query: The parsed query string object.
* hash: The hash fragment portion of the UR
  including the leading '#'.

Example:

```
const url = require ('url');
const urlString = 'https://www.example.com:80
                   path/to/resource?param1=v
                   &param2=value2 #section1';
const parsedUrl = url.parse (urlString, true);

console.log (parseUrl.protocol);
console.log (parseUrl.host);
console.log (parseUrl.pathname);

const modifiedUrl = url.resolve ('https://www.exa
                    com/base', '/relative/path')
console.log (modifiedUrl);
```

2). process :-

→ The process module in Node.js provides information and control over the current Node.js process.

→ It is a global object that can be accessed from anywhere in Node.js application without the need for an explicit 'require()' statement.

*. Methods :-

- process.exit([code]): Terminates the nodejs process. If a numerice 'code' argument is provided, it sets the exit code for the process. By convention an exit code of 0 means success and any non-zero value indicates an error or an abnormal termination.
- process.cwd(): Return the current working directory of nodejs process.
- process.memoryUsage(): Returns an object with memory usage statistics for the current process.
- process.uptime(): Returns the number of seconds the nodejs process has been running.
- process.on(event, listener): Registers event handlers for various events in the nodejs process.
- process.nextTick(callback, [...args]): Queues a callback to be executed on the next iteration of the event loop.

VALENTINE
Page No.:
Date: | |

*. Properties :-
- process.argv: An array containing the command line arguments passed to the nodejs proce
- process.env: An object containing the environ variables of the current process. It stores configuration settings.
- process.pid: Returns the process ID of proces
- process.platform: Returns the platform on which nodejs is running.
- process.stdout: A writable stream representing standard output of the process.
- process.stderr: A writable stream representin the standard error output of the process.

(3). readline :-
→ The 'readline' module in nodejs provides an interface for reading input from readable streams.
→ It is particulary useful for building CLI a interactive console applications.
→ Methods :-
* readline.createInterface(options): This method is used to create a new readline interface. It to an options object as an argument with the following properties:
- input: A readable stream from which inpu will be read.

- output: A writable stream where the output will be written.
- completer: A completer function that provides tab completion support.

* readline
* rl.question(query, callback): Displays the query to the user and waits for input. Once the user enters a complete line of input and presses Enter, the callback function is invoked with the user's response as an argument.
* rl.close(): Close the readline interface.
* rl.pause(): Pause the input stream.
* rl.resume(): Resume the input stream.
* rl.setPrompt(prompt): Sets the prompt message to be displayed when waiting for user input.

→ Properties:-
- rl.input: A readable stream representing the input stream.
- rl.output: A writable stream representing the output stream.
- rl.history: An array containing the user's input history.
- rl.cursor: An integer representing the current position of the cursor on the input li

4. fs :-

→ The fs module in Nodejs is a built-in modu that provides an interface for working with th file system.

→ Properties:

- fs.constants : An object containing constants for system-related operation, such as file access m

→ Methods :-

- fs.readFile (path, [options], callback) : Asynchronou reads the contents of a file.
- fs.readFileSync(path, [options]) : Synchronously reads the file.
- fs.writeFile (file, data, [options], callback) : Asynchronously writes data to a file. If the file does not exist, it will be created and if it exist it's content will be replaced.
- fs.writeFileSync(file, data, [options]) : Synchron writes data to a file.
- fs.appendFile (file, data, [options]) : Asynchronously appends data to a file.
- fs.unlink (path, callback) : Asynchronously remo file.
- fs.rename (oldpath, newpath, callback) : Asynchron renames a file or moves it to a different loca

Example:

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

fs.readFile("./file1.txt", (err, data) =>
{
    console.log(data);
});
fs.writeFile("./file1.txt", "Hello 1st file", () => {});
```

5. Events :-

→ Event module is a built-in module that provides the EventEmitter class. This class allows you to work with custom events in your application.

→ Properties:

- events.EventEmitter.defaultMaxListeners: The default maximum number of listeners that can be assigned to an event. If the number of listeners exceeds this value, a warning is emitted.

→ Methods:

- EventEmitter.addListener(event, listener) or EventEmitter.on(event, listener): Adds a listener function to the specified event. Whenever the event is emitted, the listener function will be called.
- EventEmitter.once(event, listener): Adds a one time listener function to the specified 'event'. The listener will be called only once, after that it will be automatically removed.

- EventEmitter.removeListener(event, listener): Re
  a listener function from the specified
- EventEmitter.removeAllListeners(event): Rem
  listeners for the specified 'event'.
- EventEmitter.listeners(events): Returns an ar
  listener functions to the specified 'event
- EventEmitter.listenerCount(emitter, event): Re
  the number of listeners for the specifie
  'event'.

Example:

```
class EventEmitter = require('events');
class MyEmitter = extends EventEmitter {}

const myEmitter = new myEmitter();

myEmitter.on('message', (msg) => {
  console.log('Received Message', msg);
});

myEmitter.emit('message', 'Hello World!')
```

6. console :-

→ The console module provides a simple
and logging interface that is similar to
console provided by web browsers. It allo
you to output messages to the console duri
execution of nodejs application.

→ Methods:-

- console.log([data],[...]): Prints the given data to the console.
- console.error([data],[...]): Prints the given data to console as an error message.
- console.warn([data],[...]): Prints the given data to the console as a warning message.
- console.info([data],[...]): Prints the given data to the console as an information message.
- console.debug([data],[...]): Prints the given data to the console as a debug message.

→ Properties:

- console.Console: A constructor function that allows you to create a custom console object with a specific output stream.

Example:

```
console.log("Hello");
console.err("This is error");
console.warn("This is warning");
```

7. buffer:-

→ The buffer module in Nodejs is a built-in module that provides a way to work with binary data in the form of a buffer.

→ Buffers are used to handle raw binary data, such as reading/writing files, working with network data and interacting with binary protocols.

→ Methods:

- Buffer.alloc(size,[fill],[encoding]): Creates a new buffer of the specified 'size' and initialize it with optional 'fill' and 'encoding'.
- Buffer.allocUnsafe(size): Creates a new buffer the specified size without initializing the contents.
- Buffer.allocUnsafeSlow(size): Same as a Buffer allocUnsafe() but slower.

→ Properties:

- buf.length: The size of the buffer in bytes that this property returns the allocated size the buffer, which may be larger than actual data.

Example:

```
const buf = Buffer.alloc(6);

buf.write('Hello!','utf-8');
```

8. Query String:-

→ Nodejs provides utilities of working with query strings.

→ It is used to represent data in URLs to encode data in HTTP requests and responses.

→ Methods:-

- querystring.parse(str[,sep[,eq[,options]]]): Parse a querystring, and return object in key value pair.

- querystring.stringify(obj[,sep[,eq[,options]]]): Converts a object into a query string.
- querystring.escape(str): Escapes a string by replacing special characters with their corresponding URL-encoded representations.
- querystring.unescape(str): Unescapes a string by replacing URL-encoded characters with their corresponding characters.

ample:

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

const queryString = 'name=Maitri&age=21';
const parsedQuery = querystring.parse(queryString)
console.log(parsedQuery);
```

http:-

- This module allows to transfer data to HTTP.
- Properties:-

- http.METHODS: Lists all the HTTP methods.
- http.STATUS_CODES: Lists HTTP status codes and description.

http.globalAgent: Maintains a queue of pending requests for each host and port.

Methods:-

http.createServer(): To return a new instance o the HTTP server class.

- http.request() : To make a http request to the server.

Example:

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

http.createServer((req,res) =>
  {
    res.end('Hello');
  }).listen(8000);
```

10. v8 :-

→ This module provides access to certain functions of the V8 JavaScript engine.

→ This module exposes some low-level features and diagnostic tools related to memory management

→ Properties :-

- v8.cacheDataVersionTag : A number of representation the V8's internal cached data version tag.

→ Methods :-

- v8.getHeapStatistics() : Return an object containing statistics about V8's heap memory usage.
- v8.getHeapSpaceStatistics() : Return an array of objects each representing statistics about a specific space
- v8.serialize(value) : Serializes a JavaScript value to a Buffer that can be used to transfer the value between Nodejs instances or even different machines.

example:

```
const v8 = require('v8');

const heapStatics = v8.getHeapStatistics();

console.log(heapStatiscs.total-heap-size);
```

11. OS :-

→ It provides a set of utility methods to interact with the opening system.

→ It provides access to information about the system's architecture, networking interfaces, environment variables and more.

→ Properties :-

- os.EOL : A string repersenting the end-of-line marker used on the current platform.

→ Methods :-

- os.arch() : Returns a string representing the CPU architecture of the OS.
- os.cpus() : Returns an array of objects containing information about the CPU cores.
- os.freemem() : Returns the amount of free system memory in bytes.
- os.hostname() : Returns the hostname of OS.

mple :

```
const os = require('os');
console.log(os.arch());
console.log(os.hostname());
```

12. zlib :-

→ It provides functionalities of compression and decompression.

→ Properties :

- zlib.constants : An object containing constants and used by the zlib module.

→ Methods of Compression :

- zlib.gzip(input[, options], callback) : Compresses the input data using the Gzip compression algorithm.
- zlib.gzipSync(input[, options]) : Synchronously compresses the 'input' data using Gzip.
- zlib.deflate(input[, options], callback) : Compresses the 'input' data using the deflate compression algorithm.

→ Methods of Decompression :

- zlib.gunzip(buffer[, options], callback) : Decompresses the Gzip compressed 'buffer'.
- zlib.createGunzip([options]) : Returns a new Gunzip transform stream.

Example :

```
const zlib = require('zlib');
zlib.gzip(originalData, (err, compressedData) => {
    console.log('Original Data:', originalData);
    console.log('Compressed Data:', compressedData.
        toString('base64'));
});
```