NAME : Rane Dhruvi

ROLL NO. : 68

DIVISION : B

SEMESTER : 7th

SUBJECT : 701 Application Development using Full Stack

ASSIGNMENT : 1 (theory)

DATE : 18 / 07 / 2023

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

Ans : 1 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 networking web applications.
- It is open source and free to use.

→ Features of Node.js

1. Extremely fast :

- It is built on Google chrome's V8 JavaScript Engine, so its library is very fast in code execution.

2. I/O is Asynchronous and event driven :

- All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data.

3. Single thread :

- Node.js follows a single threaded model with event looping.

4. Highly Scalable :

- Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way

5. No-buffering

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

6 Open Source :

- Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.

7. License :

- It's released under the MIT license

→ Execution Architecture :

- A typical web application consists of the following components :

Client : A client refers to the user who interacts with the server by sending out requests.

Server : The server is in charge of receiving client request, performing appropriate tasks, and returning results to the clients.
It serves as bridge between the front-end and the stored data, allowing clients to perform operations on the data.

Database : A database is where a web application's data is stored. Depending on the client's request, the data can be created, modified, and deleted.

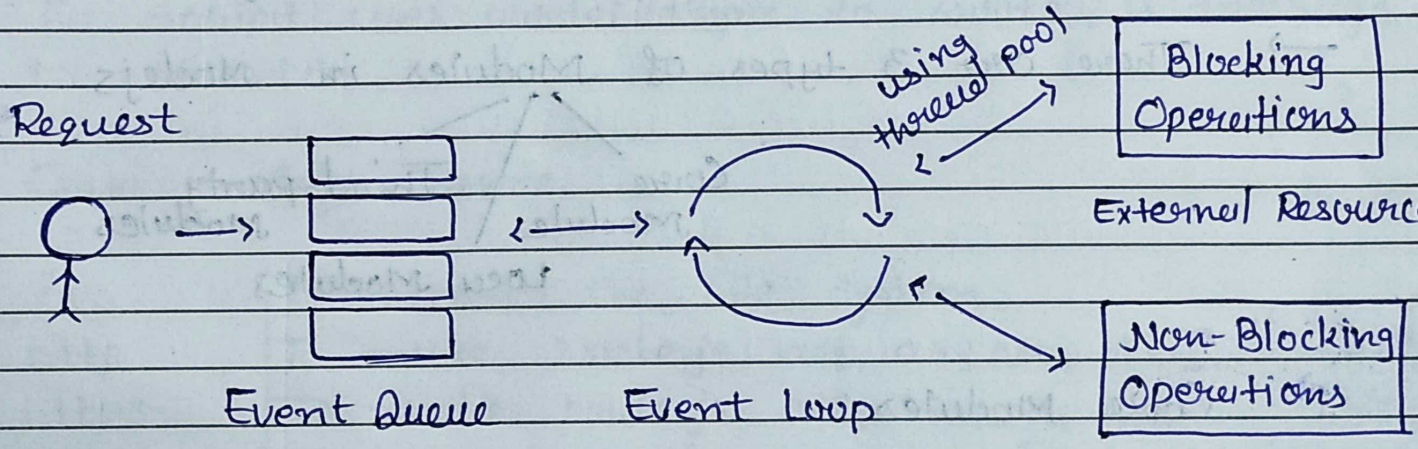
→ To manage several concurrent clients, Node.js employs a "Single Threaded Event Loop" design.
- The javascript event-based model and the Javascript callback mechanism are employed in the Node.js Processing Model.

- It employs 2 fundamental concepts:

1. Asynchronous model
2. Nonblocking of I/O operations

→ Components of the architecture:

1. Requests
2. Node.js Server
3. Event Queue
4. Thread Pool
5. Event Loop
6. External Resources

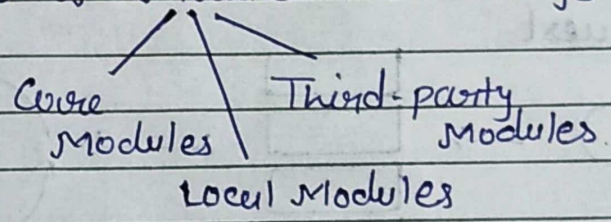


Que: 2 Note on modules with example

Ans: 2 In Node.js Application, a module can be considered as a block of code that provide a simple or complex functionality that can communicate with external application.

- Modules can be organized in a single file or a collection of multiple files / folders.
- Almost all programmers prefer modules because of their reusability throughout the application and ability to reduce the complexity of code into smaller pieces.
- Nodejs uses the commonJs module standard implementation in its module ecosystem.

→ There are 3 types of Modules in Nodejs



1 Core Modules:

- Nodejs comes with dozens of built-in modules. These built-in modules are sometimes referred to as core modules.
- The module system is built around the require function. This function is used to load a module and get access to its contents.
- require is a global variable provided to all your Node.js scripts, so you can use it anywhere you like.
- require() function will return a Javascript type depending on your module.

Syntax:

```
const module = require("module.name");
```

- You can directly use the Nodejs core module in your application without any installation.

Example:

```
const fs = require('fs'); // Import fs module
fs.writeFileSync('notes.txt', 'I love to code');
```

- The script above uses require to load in the fs module.
- This is a built-in Node.js module that provides functions you can use to manipulate the file system.
- The script uses writeFileSync to write a message to notes.txt.

→ Core modules name:

fs	To handle the file system.
http	To make Node.js act as an HTTP server.
https	To make Node.js act as an HTTPS server.
os	It provides information about the operating system.
path	To handle file paths.
cluster	To split a single Node process into multiple processes.
dns	To do DNS lookups and name resolution functions.
tls	To implement TLS and SSL protocols.
queryString	To handle URL query strings.
url	To parse URL strings.
events	To handles events.
timers	To execute a function after a given number of ms.

→ Local Modules :

- Putting all your code in a single file is not good idea.
- As you code more code, you'll want to stay organized and break your node.js app into multiple scripts that all work together.
- Exporting from files : Firstly we need to create a file called myfunc.js file.
- Now we can export javascript code written on this file using module.exports

myFunc.js file

```
const func = function (){
    Console.log ("Hello, this is modules");
}
module.exports = func;
```

Importing our files :

- Moreover, we need to import the local file into index.js

index.js file

```
Const myfunc = require ('./myFunc.js');
myFunc();
```

↳ Output :

```
Hello, this is module
```

→. Third-party modules:

- Third-party modules can be installed from the NPM (Node package Manager) available online.
- To use it, firstly we need to initiate the npm using npm init command and it will create package.json file in the root directory.
- It stores the all information about the third-party modules that we have installed as a dependency.

Installing an NPM modules:

npm install "module_name"

Example:

npm install express
npm install validator

Que : 3 Note on package with example

Ans : 3 A package contains all the files you need for a module

- Modules are javaScript libraries you can include in your project

→ Download a package

- Downloading a package is very easy.
- Open the command line interface and tell NPM to download the package you want.
- I want to download a package called "upper-case":

```
C:\Users\Your name> npm install upper-case
```

- Now you have downloaded and installed your first package!
- 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

→ Using a Package:

- Once the package is installed, it is ready to use.
- Include the node-static package

```
Const static = require("Node-static");
var http = require("http");
var static = require("Node-static");
Const fileserver = static.Server('./Test');
http.createServer((req, res) => {
    res.writeHead(200,{'content-type':'text/html'});
    res.Fileserver.serve('ABC.html', res, {},
                                    req, res);
    res.end();
}).listen(8080);
```

Que : 3 Note on package with example

Ans : 3 A package contains all the files you need for a module

- Modules are javaScript libraries you can include in your project.

→ Download a package

- Downloading a package is very easy.
- Open the command line interface and tell NPM to download the package you want.
- I want to download a package called "upper-case":

```
C:\Users\Your name> npm install upper-case
```

- Now you have downloaded and installed your first package!
- 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

→ Using a Package:

- Once the package is installed, it is ready to use.
- Include the node-static package

```
Const static = require ("Node-static");
var http = require ("http");
var static = require ("Node-static");
Const fileserver = static.Server('./Test');
http.createServer ((req, res) => {
    res.writeHead (200, {'content-type': 'text/html'});
    res.Fileserver.serve ('ABC.html', res, {},
                          req, res);
    res.end ();
}).listen (8080);
```

Que : 4 use of package.json and package-lock.json.

Ans : 4 In Node.js, 'package.json' and 'package-lock.json' are fundamental files used to manage dependencies and configurations for a project.

- They play a crucial role in ensuring consistent and reliable behaviour across different environments and when collaborating with other developer.

→ Package.json :

- The package.json file is a json (Javascript Object Notation) file that serves as a manifest for node.js project or package.
- It contains essential metadata and configuration information including :

name : The name of the package or project.
version : The version of the package or project.
description : A brief description of the package or project.
main : The entry point file for the application.
dependencies : A list of packages that the project depends on only during development.
Scripts : Custom scripts that can be run using 'npm run <Script-name>'.
author : The author of the package or project.
license : The license under which the package or project is distributed.

→ Use of package.json :

- Dependency Management : The dependencies and ~~development~~ devDependencies Sections list the packages required for the project to work correctly.

→ Script execution : The script section allows you to define custom scripts that can be executed using

'npm run <Script Name> ;

- This makes it easier to manage common tasks associated with the project.

→ Version Control : The package.json file is typically included in version control. This ensures that the package's configuration and dependencies are consistent accross different development environment and when collobirating with other developers.

→ Package-lock.json :

- The package-lock.json file is another json file that npm automatically generates when you run 'npm install'

→ Use of Package-lock.json :

→ Dependency version locking :

- The primary purpose of package-lock.json is to lock the version of each installed dependency.
- It records the specific versions of each package and its transitive depencies that were installed when 'npm install' was executed.
- This guarantees that anyone else who installs the project using the same package-lock.json will get exactly the same versions of all depencies, ensuring consistency accross different environment.

→ Faster and safer installs :

- By using it, npm can avoid resolving dependencies and their versions every time 'npm install' is run. Instead, it can quickly install the exact versions specified in the package-lock.json which makes installation faster and more reliable

→ Security :

- It also helps enhance security by preventing potential security risks that might arise from using different versions of dependencies.
- Since the exact versions are locked, any security patches applied to these version will be automatic used when the project is installed.

Que : 5 NPM introduction and commands with use

Ans : 5 NPM, which stands for node package manager, is a package manager for the Javascript programming language.

- It comes bundled with Node.js, a runtime environment for executing Javascript code server-side.
- NPM is widely used in the web development community to install, manage and share open-source Javascript libraries and tools.
- Here's a brief intro to npm and some commonly used commands:

1. Intitialize a new project:

```
npm init.
```

To start a new project, you first need to initialize a 'package.json' file.

- This command will prompt you to answer some questions to set up your 'package.json' file. Or you can use 'npm init -y' to accept all default.

2. Install packages:

You can use npm to install packages from them npm registry and add them to our project's dependencies.

```
npm install express.
```

- This will download the express package and add it to the dependencies section of our package.json file.

3. Install packages as development dependencies

Sometimes we need packages only while the development process, such as testing libraries or build tools.

- For that we use --save-dev.

npm install package --save-dev

4. Install packages globally:

Some packages provide command-line utilities that you want to use across different projects.

- So we can install them globally -g.

npm install -g nodemon.

5. Update package:

To update package to latest or updated version to our project we can utilize of update.

npm update

6. Uninstalled packages:

To remove a package from our project we use uninstall command.

npm uninstall node-static

7. List installed packages.

To see a list of packages installed in our project we use ls command.

npm ls

8. NPM run script :

In package.json file we can define custom script that we can run using npm run.

npm run start

9. Search for packages :

We can search for packages in the npm registry.

npm search package-name

10. Publish your package :

If we've created a reusable library or tool, we can publish it to the npm register for others to use.

npm publish

Que: 6 Describe use and working of following Node.js packages, important properties and methods and relevant programs

Ans: 6 URL :

- The URL module is built-in Node.js module that provides utilities for working with URLs.
- It allows us to parse, formate and manipulate URLs

→ Methods :

URL.parse (URLString, purseQueryString [..,]);

This method purse a URL string and returns an object containing its various components

→ Example :

```
const url = require ('url');
const urlString = "https://www.google.com";
const purseURL = url.parse (urlstring, true);
```

↳ output :

```
{
    protocol : 'https:',
    slashes  : true,
    auth     : null,
    host     : www
    port     : null,
}
```

→ url formate (urlObject);

This method takes a URL object and converts it back into a formetted URL string.

```
const url = require('url')
const obj = {
    protocol : 'https:',
    host     : 'www.example.com',
    pathname : '/path/to/resource',
    query    : { key : value }
};
const formatedurl = url.format(obj);
console.log(formatedurl);
```

↳ output :

https://www.example.com/path/to/resource
key=value

→ url.resolve(from, to):

This method resolves a relative URL to an ~~abs~~ absolute URL.

→ Properties :

protocol : "http:", "https:", etc.
slashes : "//"
auth : 'username:password'
host : 'www.google.com'
port : Port of url
query : an object representation of the parsed query string.

2. Process :

- The process is built-in global object in Node.js that provides information and control over the current Node.js process.
- It allows you to interact with the process running your node.js application, access environment variables and handle events related to the process.

→ Process.argv :

This property contains an array of command line arguments passed to the Node.js application.

```
node app.js arg1 arg2
console.log(process.argv);
```

→ Process.env :

This property contains an object representing the user environment.

process.env.Node_ENV

→ Process.pid :

This property holds the process id of the current node.js process.

→ process.cwd() :

This method return the current working directory.

→ pm2 : pm2 is process manager for node.js applications.

- Starting an application : pm2 start app.js
- Listing Running application : pm2 stop app name
- Restart an application : pm2 restart app name
- View logs : pm2 logs app name

3. readline :

- The readline module in node.js provides an interface for reading input from a readable stream in an interactive manner

→ Properties :

readline createInterface (options) :

- This method creates a new Interface instance
- It takes an options object as an argument, which specifies the input and output stream to use.

→ Methods :

rl.question (query, callback) :

- This method prompts the user with the 'query' and waits for user input.

Ex: rl.question ('what is you name ?', (name) =>
{
console.log (`Hello, ${name} !`) ;
rl.close();
})

rl.setprompt (prompt) :

- This method sets the prompt that is shown to the user.

Ex: rl.setprompt('enter name:');
rl.prompt();

rl.pause();

- This method pauses the readline interface.

Ex: rl.pause();

rl.resume();

- This method resumes the readline interface.

3. fs:

- The fs module in node.js stands for the system.
- It provides an interface for working with the file system, allowing you to read, write and the manipulate files and directories on your computer or server.

→ Read file:

fs.readfile(path[, options], callback) reads the content of the file.

Ex: const fs = require('fs');
fs.readfile('ABC.txt', 'UTF-8', (err, data)=>{
if(err) throw err;
console.log(data);
});

fs.readFileSync(path[, options]) produce the content of a file synchronously.

Ex: console.log(fs.readFileSync('ABC.txt', 'UTF-8'));

→ WriteFile :

fs.writeFile(file, data[, options], callback);

- writes data to a file asynchronously. if the file does not exist, it creates a new one.

Ex. : fs.writeFile('ABC.txt', 'Hello World',
(err) => {
if (err) throw err;
console.log('file written successfully');
});

4. Events :

- In node.js the 'events' module is a core module that provides a mechanism to handle and emit events.

→ EventEmitter class :

It is a core of a events module. it provides methods to handle and emit events.

→ Event handling :

This method is used to register an event listen for a specific event.

myEmitter.on('event', () => {
console.log('event emitted');
});

Ex : const EventEmitter = require('events');
class MyClass extends EventsEmitter {
constructor() {
super();
}
}

```
const obj = new myClass()
obj.on('event', () => {
    console.log('custome event');
});
obj.emit('event');
```

5. Console :

- The console is a buit-in object in node.js and web browsers that provides a simple debugging and logging mechanism
- It allows you to interact with the standard output and standard error streams to display messages, debug information and errors during the execution of your code.

→. Methods :

console.log()

- prints the provided data and objects to the console separed by spaces.
- It automatically converts the values to string

Ex:
```
const name = "dhruvi";
const rno = "68";
console.log("name:", name, "rno:", rno);
```

Console.error():

- Prints to the standard error stream instead of standard output.

Ex: console.error("this is error");

console.warn():

- Prints to standard error stream, but it is often

used for less server warning or alerts.

Ex: consol.warn("warning : low risk");

Console.clear()

- It is clear the console.

6. Buffer :

- The buffer class is a buit-in global object that provides a way to handle binary data.
- A buffure is like an array of integers but each element represents a single byte of data.

- creating Buffer:

```
Buffer.alloc(5);
Buffer.from([65, 66, 67]); // create buffer.
```

→ Common use cases :

- Reading from and writing to streams, files and sockets.
- Handling binary data in networking protocols, encryption and compression.
- parsing and serializing binary data formates like BSON or protocol Buffers.

7. queryString :

- The querystring is a part of the URL that comes after the question mark '?'. It is used to send data to the server as key-value pairs.

→ Parsing QueryStrings :

```
queryString.parse([str[, sep[, eq[, options]]]]);
```

→ creating query string :

```
querystring.stringify([obj[, sep[, eq[, options]]]]);
```

- The queryString module in nodejs is helpful when dealing with URLs and processing query parameters in web applications and APIs.

7. http :

- The http module in nodejs is a core module that provides functionality to create HTTP servers and make HTTP requests.
- It allows you to handling incomming http request and serve responses, making it a fundamental building block for building web applications and APIs in node.js

→ creating an HTTP server :

```
const http = require("http");
const server = http.createServer((req, res) =>
{
    res.end("Hell world");
}).listen(8000);
```

8. V8 :

- V8 is an open-source js engine developed by google, It is written in c++ and is primarily used to execute Javascript code in web browser and server-side environments

- V8 is at the core of many popular js environments including google chrome, Node.js and several other runtime environments

→ Features :

1. Just-in-time (JIT) Compilation :

- V8 uses a Just-in-time compilation technique to optimize the execution of JS code.
- When a JS program is run, V8 first interprets the code and then compiles parts of it into highly optimized machine code for better performance.
- This adaptive compilation helps improve the speed of executing JS

2. Garbage Collection :

- V8 implements a garbage collector to manage memory allocation and deallocation for JS objects

3. Multi-platform support :

- V8 is designed to be portable and run on various platforms, including windows, macOS, Linux and mobile operating systems like Android and iOS

4. Supports for ECMA Script standards :

- V8 is compliant with the ECMA Script standard, which defines the syntax and semantics of the JS language.

5. Isolation and sandboxing :

- V8 provides an isolation mechanism that allows running multiple JS contexts independently within the same process

6 Profilling and Debugging:

- V8 includes tools for profilling and debugging Js code. Developers can use tools like chrome DevTools and Node.js inspector to inspect memory usage, CPU performance and debug Js code

7. Embedding in custom applications:

- V8 is designed to be embedded, which means it can be integrated into custom application to allow these applications to execute Js code.

9. os :

- The 'os' module in Node.js is a core module that provides operating system related utilities.

os.platform():

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

↳ output: win32

10. Zlib :

- Zlib module in node.js is a core module that provides compression and decompression functionalities using the Zlib library.

→ Compression:

Zlib.default

zlib.deflate(buf[, options], callback)

```
const zlib = require('zlib');
const input = Buffer.from("Hello, zlib compress");
zlib.deflate(input, (err, compressed) => {
    if (err) {
        console.log(err);
    }
});
```

→ zlib.inflateSync(buf[, options])

- It decompressed the data.

```
const zlib = require('zlib');
const compressed = ..... // compressed data
const decompressed = zlib.inflateSync(
                        compressed);
console.log('decompressed data', decompressed);
```

- In node.js the 'zlib' module is a powerful tool for compressing and decompressing data using various algorithms and formats.