Assignment :- 1

Name :- Patel Pooja Nareshbhai

Roll NO :- 054

Subject :- Full stack

Semester :- 7th sem

Subject Code :- 701

Date :- 21/07/23

Submission Date :- 24/07/23

Que: 1 Nodejs : Introduction, features, execution architecture.

→ Introduction :-

→ Nodejs is an open source and cross platform runtime environment for executing javascript code outside a browser.

→ Nodejs is not a framework and it's not programming language.

→ Nodejs is use for building back-end service like APIs like web APP or mobile APP.

Features of Nodejs :-

→ it's easy to get started and can be used for prototyping and agile development.
→ it provides fast and highly scalable services.
→ It uses Javascript everywhere so it's easy for a javascript programmer to build back-end services using Nodejs
→ Source code cleaner and consistent.
→ Large ecosystem for open source library
→ It has Asynchronous or Non-blocking nature.

* Execution Architecture of Node.js :-

1. event Loop :-

The event loop as the core of Node.js asynchronous and non-blocking architecture. It continuously runs and waits for events to occur. when an event is detected. it triggers the corresponding callback function and the event loop moves on to the next event.

2. callbacks :-

callbackes are a fundamental part of Node.js development when asynchronous operation complete they trigger callback functions, allowing developers to handle the results or errors accordingly. callbacks are passed as arguments to asynchronous functions making it possible to perform action after specific operation complete.

3. Event Emitters :-

Event emitters are objects in node.js that can emit named events and attach listeners to those events. They form the basis of many core Node.js modules and are widely used for handling various asynchronous operations.

4. Non-blocking I/O :-

Nodejs is utilizes non-blocking I/O operations, enabling it to handle multiple requests simultaneously without waiting for one operation to complete before moving to the next. This approach makes Nodejs highly efficient, especially in applications that require frequent I/O operations.

5. Threads and concurrency :-

Though Nodejs runs on a single thread it employees a thread pool for executing certain I/O operations like file system access. This allows Nodejs to take advantages of multi-core processors while still maintaing its single-threaded, event-driven nature

6. modules and NPM :-

Nodejs encourage the use of modules, which are independent unit of functionality that can be reused across applications. These modules are managed using NPM which simplifies dependency management and makes it easy to integrate third-party libraries

Que: 2 Note on modules with example.

→ In Nodejs 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 Programmer Prefer modules because of their reusability throughout the application and ability to reduce the complexity of code into smaller pieces.

Types of modules :-

1. Core modules
2. Local modules
3. Third Party modules

1. core modules :-

Nodejs comes with dozens of built in modules. These builtin modules are sometimes referred to as core modules. The modules 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 call your Nodejs scripts so you can use it anyahere you like.

Syntax :-

```
const module = require("Name_of_import_module");
```

Example :-

```
Const fs = require('fs');
fs.writeFileSync('note.txt', 'I love code');
```

List of some Nodejs core modules :-

1. fs :-
   TO handle the file system

2. http :-
   TO make nodejs act as an http server.

3. https :-
   TO make Nodejs act as an https server.

4. os :-
   it provides information about os.

5. Path :-
   TO handle file Paths.

2. Local modules :-

Putting all your code in a single files is not a good idea. As you add more code, you will want to stay organized and break your Node.js app into multiple scripts that all work together. For that purpose we can create local modules in our application.

Example :-

* Utils.js :-

```
const tests = function () {
    console.log("Hello world");
}
module.exports = tests;
```

importing your own files :-

We need to import the local file into index.js

* index.js :-

```
const utility = require('./utils.js');
utility();
```

3. Third-Party Modules :-

This modules can be installed from the NPM (Node Package manager) available online.

Firstly we need to initialize the npm using the npm init command

before npm can be used. it create a package.json file in the root directory and it stores all the information about the third-party module that we have installed as a dependency.

Syntax :

```
npm install "module-name"
```

Example :

```
npm install express
npm install validator.
```

index.js :

```
Const validator = require("validator");

console.log ("URL is " + validator.URL
("https://google.com"));

console.log ("Email is " + validator.
isEmail("abc@gmail.com"));
```

Que:3 Note on Package with example.

→ *Npm is the default Package manager for Nodejs and is written entirely in javascript.

→ Packages is a collections of different different module.

→ Npm manages all Packages and modules for Nodejs and consists of command line client npm.

→ A Package contains all the files needed for a modue and modules are the javascript libraries that can be included in Node Project according to the requirement of the Project.

→ Npm can install all the dependencies of a project through the Package.json File.

→ It can also update and uninstall Packages.

Syntax :-

npm install Package_name.

Example :- npm install express

Que: 4 Use of Package.json and Package-lock.json.

→ Package.json and Package-lock.json are two essential files used in nodejs projects to manage dependencies and ensure consistent installations across different environment.

* Package.json :-

→ This file is a JSON file that resides in the root of a Nodejs Project. It serves as the manifest file for the project and contains essential information, the list of dependencies required for the Project to run.

→ The dependencies section lists the Packages required for the project to run in Production.

→ The devDependencies section lists Package required for development purpose

→ The scripts section provides shortcuts to execute various commands.

* Package-lock.json :-

→ The Package-lock.json file was introduced in npm version 5 and serves as a lockfile for npm dependencies. it is automatically generated and should be committed to version control file along with the Package.json file.

→ The Primary Purpose of Package-lock.json is ensure that the exact versions of the installed dependencies are consistent across different enviornments.

→ exact version of all the installed Packages, including the nested dependencies.

→ Integrity hashes for each Package with guarantee the Package's content hasn't been altered.

→ By having the Package-lock.json you ensure that anyone else working on the Project will have the same versions of dependencies installed which reduces the chances of compatiblity issues and keeps the project stable and Predictable.

Que:5 Nodejs Packages

→ Node.js is a popular javascript runtime that allows developers to execute javascript code outside of a web browser, enabling server-side development and building various application.

Nodejs has a vast ecosystem of packages available through the Node Package manager. These Packages provide reusable code and functionality to simplify development tasks.

Node.js Packages :-

1. Express :-
A fast and minimalist web application framework that simplifies building web servers and APIs.

2. Axios :-
A popular http client for making API requests from Node.js applications.

3. Lodash :-
A utility library that provides a wide range of useful functions for manipulating and working with arrays, objects, strings, etc.

4. Socket.io :-
A real-time web socket library that enables bidirectional communication between clients and servers.

5. Mongoose :-
An object Data modeling library for mongoDB providing a convenient way to interact with the database.

6. dotenv :-
A package for loading environment variables from a '.env' file into your Node.js application.

7. Body-Parser :-
A middleware to parse incoming request bodies in a middleware-friendly way.

8. Nodemailer :-
A module to send emails from Node.js applications.

9. morgan :-
A request logger middleware to log http requests in Node.js applications.

Syntax :- npm install Package-name

Que:6 npm introduction and commands with its use.

→ NPM which stands for Node Package manager is a command-line tool and Package manager for javascript Programming. It comes bundled with Nodejs and allows developers to easily manage dependencies, install Package and share their own Packages with others. It is widely used in the Nodejs ecosystem and is an essential tool for javascript developers.

Some Common npm Commands :-

1. npm init :-
Initializes a new nodejs Project in the current directory. It Prompts you to Provide some basic information about your Project, such as the Package name, version, description, entry point, author and license. This command generates a Package.json file that contains the Projects configuration and dependencies.

2. npm install <Package-name> :-
Installs a Package and its dependencies locally in your Project. The Package is downloaded from

the npm registry and saved in the node_modules directory of your Project. The dependencies are also recorded in the Package.json file.

3. npm install <Package-name> --save<dev>
or npm install <Package-name -D :-

installs a Package as a development depency. This means the Package is required for development Purposes such as testing or building but is not nedded for the production version of the Project.

4. npm install :-
installs all the Packages listed in the Package.json file. It reads the Package.json file and installs all the Packages defined under 'dependencies' and devdependencies.

5. npm uninstall <Package-name> :-
Removes a Package from the Projects node_modules directory and updates the Package.json file accordingly

6. npm run <script-name> :-
Executes custom scripts defined in the scripts sections of Package.json. These Script can be used for various Purpose like running tests, starting the building Project.

Que: 7 Describe use and working of following Nodejs Packages. Important Properties and methods and relevant Programs.

→

1. url :-

-> The url module is used for URL Parsing and formatting. It provides utilities for working with URLS and allows you to extract information from URLS.

Important Properties and methods :-

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

This method takes a URL string as input and return an object with various properties like protocol, hostname, port, path, query etc.

2. url.format(urlObject) :-

This method takes a URL object and return a formatted URL string

Program :-

```
const url = require('url');
const urlString = 'http://www.google.com';
const ParsedURL = url.parse(urlstring, true);
console.log(ParsedUrl.protocol);
console.log(ParsedUrl.hostname);
console.log(ParsedUrl.query);
```

2. Process :-

The Process module provides information and control over the current Node.js Process. It allows you to interact with the OS and environment variable.

Properties and methods :-

1. Process.argV :

An array containing the command line argument passed to the Node.js Process.

2. Process.env :

An Object containing the user environment.

3. Process.cwd() :

Returns the current working directory.

4. Process.exit([code]) :-

Terminates the Node.js Process with an optional exit code.

Program :

```
console.log( Process.argv);
console.log( Process.env.NODE_ENV);
console.log (Process.cwd());
Process.exit(0);
```

3. Pm2 (external Package) :-

Pm2 is an external Node.js package that provides advanced process management for application. It allows you to manage application lifecycle, clustering, monitoring and automatic restarts.

Program :

```
Pm2 start app.js
Pm2 list
Pm2 stop app.js
Pm2 restart all
```

4. readline :-

The readline module provides an interface for reading data from a Readable stream.

Program :

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

rl.question('How are you ?');
console.log('fine');
rl.close();

});
```

4. fs :-

The fs module allows you to work with the file system, read/write files and perform file-related operations.

Program :-

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

fs.readFile('file.txt', 'utf8', (err, data) => {
    if(err) throw err;
    console.log(data);
});

fs.writeFile('file.txt', 'Hello', (err) => {
    if(err) throw err;

});
```

5. events :-

The events module is the core of the Node.js event driven architecture. It allows Objects to emit and listen for custom events.

Progm :-

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmitter.on('customEvent', (arg) => {
    console.log('argument :-', arg);
});

myEmitter.emit('customEvent', 'Hello');
```

6. Console :-

The console module provides a simple debugging console that is similar to the console mechnism provided by web Browser.

Program :

```
console.log('Hello, world');
```

7. buffer :

The buffer module provides a way to handle binary data. It is used for working with binary streams and raw data

Program:-

```
Const buf = Buffer.from('Hello, world', utf8);
console.log(buf);
```

8. querystring :

The querystring module is used for Parsing and formatting URL query strings.

Program :-

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

const querystring = 'name = Pooja & age = 21';
const ParsedQuery = querstring.Parse(
        querystring);

console.log(Parsed Query);

const ConstructedQuery = querystring.
        stringify(ParsedQuery);

console.log(constructedQuery);
```

9. HTTP :-

Node.js has buitin module called http which allows Node.js to transfer data over the Hyper text transfer Protocol.

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

10. V8 :-

V8 is the name of javascript engine that powers google chrome. It's the thing that takes our javascript and executes it while browsing with chrome. V8 Provides the runtime environment in which javascript execute.

11. OS :-

The OS module Provides information about the computer Operating system.

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

method :

arch(), constants, freemem(), hostname(), Platform(), tmpdir(), type(), userinfo().

12. zlib :-

The node:zlib module Provides compression functionality implemented using GzIP, Deflate/Inflate and Brotli.

compression and decompression are built around the Node.js.

Syntax :

```
Const zlib = require('node:zlib');
```