Name - Tailor Kruti P.

Roll No - 76

Sub - ~~FullStack~~ Application development using full stack.

Sem - 7th

Assignment - 1

Q-1 Node.js : Introduction, features & execution architecture?

→ Node.js is an open-source server side runtime environment built on chrome's V8 Javascript engine.
→ It provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly server-side application using Javascript.
→ Node.js can be used to build different types of applications such as command line application, web application, real time chat application, REST API server etc.
→ However, it is mainly used to build network program like web server, similar to PHP, Java or Asp.net.
→ There are few fetures of Nodejs are given below:

1. Single Thread

→ Node.js operates on a single thread. It is Based on the "Single Thread Event loop Model" architetur which can handle mutiple client requests.
→ A single thread executes the main event loop. Still in the background the input-output work is performed on seperate threads. As Node API's I/O-operation is asynchronous to accommodate the Event loop.
→ Event loop is what allows node.js to perform all the non-blocking operations.

2. Asynchronous

→ Node.js is asynchronous by default. i.e. it operates non-blocking. When a client request arrives a single thread handles the request; it checks if the request involves any database interaction.
→ If it does not we process the request, and the server return the respons to the client. The thread is ready to handle the next request.

3. Event - Driven

→ The event-driven concept is similer to the concept of a callback function in asynchronous programming. The only difference is that the callback function executes once the asynchronous function return its result, triggering events on it associated event handler.
→ Node provides a module called "event" which consists of an "EventEmitter" class that allows us to implement event-driven programming. An event handler is a function called when an Event is trigger
→ EventEmitter consits of various method's one of which is the emit(), that fire an event.
→ Event driven programming makes node pretty fast and gives high performance.

4. Open Source

→ Node.js is open source, which means it is free to use. We can install Node.js from https://nodejs.org/en/ according to our platform.

## 5. Performance

→ Google chrome's v8 Javascript engine is the fundamental of node.js enabling faster code execution. The engine compiles the Javascript code into machine code which makes our code easier and faster to implement effectively.

## 6. Highly Scaleable

→ Node.js applications are highly scaleable as they operate asynchronously.
→ Node.js works on a single thread, ~~start~~ where when a single request arrives, it start processing it and is ready to handle the next request. Once we prepare the response, we send it back to the client

## 7. Node Package Manager (NPM)

→ As we know, the node package manager is a package manager for Node Javascript runtime environment and is a recommended feature of the node.js installer.
→ It is the world's largest online repository. It also looks after the management of the local dependencies of our project. It has nearly 50000-80000 packages in its public online repository.

## 8. No Buffering

→ Node.js applications never buffer any data. These applications simply output the data in chunks.

9. License

→ Node.js is licensed under MIT license.

⇒ Execution architecture of Node.js

Q-2 Node on Modules with Example.

→ Consider modules to be the same as Javascript libraries.
→ A set of functions you want to include in your application.
→ Each module in Node.js has its own context, so it can't interfere with other modules or pollute global scope.
→ Also, each module can be placed in a separate .js file under a separate folder.
→ A module in Node.js is a grouping within one or more scripts that can be utilized frequently throughout much of the Node.js app & may be simple or complex in nature.

Ex
```
var http = require('http');
var dt = require('./myfirst module');
http.createServer(function (req, res)
{
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write("The date & time is currently: "
    + dt.myDateTime());
    res.End();
}).listin(8080);
```

Q-3 Note on : package with Example.

→ A package in node.js contains all the files you need for a module.
→ Modules are Javascript library you can include in you project.

→ To make development in JS easier many Node.js packages are available that help in handling errors formatting code, deleting files, debugging code, etc.

Ex
- 'upper case' package.

```
var http = require('http');
var uc = require('upper-case');
http.createServer(function(req, res){
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(uc.upperCase("Hello World"));
    res.end();
}).listen(8080);
```

⟹ Output : HELLO WORLD

## Q-4 Use of package.json & package-lock.json.

→ Package.json & package-lock.json are two important files used in Node.js projects to manage dependencies & insure consistent package installations.

→ They play a significant role in creating a stable & reproducible development environment.

* Package.json :-

→ It's a metadata file for your Node.js project.

→ It contains information about project, such as the project name, version, description, entry points, script & most imp, the dependencies required for the project to function correctly.

Ex package.json file

```
Json
{
  "name" : "my-node-project",
  "version" : "1.0.0",
  "des." : "A simple Node.js project",
  "main" : "index.js",
  "script" : {
      "start" : "node index.js",
      "test" : "Test" },
  "dependencies" : {
          "express" : "^4.17.1",
          "leader" : "^4.19.2"
  },
  "devDependencies" : {
      "Test" : "^27.0.0"
  }
}
```

⇒ Package-lock.json

→ Automatically generated by NPM
→ When you run "npm install" to install the project dependencies.
→ It ensures that the ~~install~~ is installition of dependencies & reproducible "across different environment.

→ It locks down the specific versions of all the installed package along with their sub-dependencies.

→ Fileds small excerpt from a 'package-lock.json file

- Json

```
{
  "name" : "my-node-project",
  "version" : "1.0.0",
  "lockfile version" : 2,
  "requires" : "true",
  "dependencies" : {
      "express" : {
          "vesion" : "4.17.1",
          "resolved" : "https://registry.npm.js."
          "integrity" : "sha512-mG1VBHy2..."
          "dev" : true,
          "requires" : {
              "accepts" : "1.3.7",
              "array-flatten" : "1.1.1",
              "body-parser" : "1.19.0",
              //(other dependeies)
          }
      },
      //(other dependeveies)
  }
}
```

→ The "Package.json" file defines the rules required to run your application and install dependeneies

→ Package-lock.json files holds detailed information on all the dependeneies installed based on the package.json rules.

Q-5 Node.js packages.

→ A package in Node.js contain all the files you need for a module.
→ Modules are Js libraries you can include in your project
→ Download a package

→ Downloading a package is very easy.
→ Open the command line interface & tell NPM to download the package you want.
→ 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
→ Once package is installed, it is ready to use.
→ Include the package the same way you include any other modules.
→ Create Node.js file. Done

Q-6 NPM introduction & commands with its use.

→ NPM is package manager for the node Javascript platform.
→ Use npm install <package-name> to install a new package.
→ NPM is known as the world's largest software register
→ Its puts modules in place so that node can find them, & manages dependency conflicts intelligently.
→ Most commonly, it is used to publish, discover, install & develop node programs.

1. npm install :-

→ npm comes bundled with node.js so when you install node.js on your system, npm is automatically installed alongside it.

2. Package.json :-

→ The package.json file is the heart of any node.js project. It contains metadata about the project.

3. Installing packages :-

→ To install a package, use the npm install cmd followed by package name.
ex npm install lodash.

4. Dependencies :-

→ when you install package using npm, it automatically adds the package as a dependency in the package.json file.

5. Uninstalling packages :-

→ To remove package from your project use the 'npm uninstall' command followed by package name.
ex npm uninstall lodash.

6. Update package :-

→ To update packages to the latest version, you can use the 'npm update' command followed by the package name.

7. Execute scripts :-

→ In the package.json file, you can define scripts that can be executed using the 'npm run' command.
→ You can define a script called start to run your application, then use 'npm run start' to start your node.js application.

8. Search for packages :-

→ You can search for packages on the npm registry using 'npm search' followed by a keyword.
eg npm search logging.

9. Publishing package :-

→ If you have developed node.js module or library and want to share it with the community, you can publish it to the npm registry using 'npm publish' command.

→ npm plays a crucial role in the node.js making it easier for developers to manage dependencies and share code, fostering collaboration and code reuse among the node.js community.

Q-7 Describe use & working of following node.js packages important properties and methods & relevent programs.

url, process, ipm2 (External package)
readline, fs, event, console, buffer
querystring, http, v8, os, zlib.

(i) url :-

Use - The url package provides utilities for URL resolution and passing.

working - It allows you to manipulate URLs, parse their component and resolve relative URLs into absolute URLs.

properie & methods - url.parse
Url.formed
Url.resolve.

Ex const url = require('url');
const urlstring = "www.google.com";

const parseurl = url.parse(urlstring, true);
console.log(parseurl);

const formaturl = url.formed(parseurl);
console.log(formateurl);

(ii) Process :-

Use - The process object provides information & control over the current node.js process.

working - It allows interaction with the process, including listening for signals, accessing env variables, terminating processes.

Properties & Methods :- process.argv
process.env
process.exit(code)

Q console.log(process.argv);
console.log("first argument", Process.argv[2])

(iii) Readline :-

Use - The readline module provide an interface for reading data from a readable stream.

working - It allows you to read data line by line making it useful for building cmdline interface.

properties & methods :- readline.createInterface
rl.on('line', callback)
rl.close();

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

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

(iv) fs :-

use :- The fs module provides file system relate functionality, allowing you to read from and write to files.

Working - It interacts with the file system, performing operation like reading writing, deleting, renaming files.

properties & Methods :- fs.readfile
fs.writefile
fs.unlink.

ex
```
const fs = require('fs');
fs.writefile('myfile.txt', 'Hello', (err) => {
    if (err)
    {
        console.error(err);
    }
    else
    {
```

```
    console.log("written");
});
```

(v) events :-

use = The event module provide an event-driven architecture for building scalable application with event emitters & listeners.

Working = It allows you to create custom events & register event listeners to handle those events.

Properties & Methods = events.Emitters.
eventEmiter.on(eventname, listener)
event Emitter.emit(eventname, [args])

eg
```
const event = require('events');
class myemit extends EventEmitter {}

const myemitter = new myEmitter();
myemitter.on('greet', (name) => {
    console.log("Hello', ${name}!');
});

myemit.emit('greet', 'John');
```

(vi) Console :-

Use - The console module provides a simple debugging console similar to the browser's console object.

working - It allows you to log information, errors & writting during the development phase.

Methods - Console.log()
Console.warn()
Console.error()

Ex Console.log("Hello");
Console.warn("warming");
Console.error("errors");

(vii) Buffer :-

Use - The buffer module provides a way to handle binary data.

working - It allows you to work with streams of binary data, such as reading & writting files, network communications etc.

property & Method - Buffer.allow(Size)
Buffer.from(data)
buf.toString([encoding])

Ex
```
const buf1 = Buffer.alloc(8);
console.log(buf1);

const buf2 = buffer.from('Hello');
console.log(Buf2);

Const buf3 = buff.toString('utf-8');
Console.log(buf3);
```

viii) Query String :-

Use - This module provides utilities for passing and formating URL query string.

working - It allows you to work with the query component of a URL, which is commonly used to pass data in HTTP request & response.

Methods - queryString.parse(str)
queryString.Stringify(obj)

Ex
```
const qs = require('querystring');
const qs = 'name=kriti';
const pq = querystring.parse(qs);
console.log(pq

const obj = {name: 'kriti'};
const FQ = queryString.Stringify(obj);
console.log(FQ);
```

(ix) http :-

use :- The http module provides functionality for http server and making http requests.

working - It allows you to build web servers. handle incoming request and send HTTP request to other servers.

properties & Methods - http.createServer()
server.listen(port, [host], callback)
http.get(url, callback);

ex
```
const http = require('http');
const server = http.createServer((req, res) => {
  res.end('Hello');
});

server.listen(8000, () => {
  console.log('listening');
});
```

(x) v8 :-

use - The v8 module provides access to the v8 Javascript engine's Internal feature.

working - It allows you to gather information about memory use, CPU, usage, other low-level v8 engine information.

Properties & Methods - v8.getHeapStatistics();
v8.cachedDataVersionTag();

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

(xi) OS :-

Use - The os module provides operating system related utility functions.

working - It allows you to access information about the OS, such as network interface CPU architecture and system uptime.

properties & Methods - os.platform()
os.cpus()
os.totalmem()

ex
```
const os = require('os');
console.log('platform', os.platform());
console.log('cpu:', os.cpus());
console.log('memory:', os.totalmem());
```

(xii) zlib :-

Use :- The zlib module provides compression & decompression functionalities using Gzip, Deflate & Brotli algorithms.

working - It allows you to compress data for efficient storage and transmission and decompress it when need.

properties & Method - zlib.gzip( input, callback)
zlib.gunzip( input, callback)
zlib.deflate(input, callback)

```
const zlib = require('zlib');
const data = 'Hello H';
zlib.gzip( data, (err, compressedData) => {
  if(err) {
    console.error(err)
  } else {
    console.log(compressed data);
    zlib.gunzip(compressdata, (err, ddata) => {
      if(err) {
        console.error(err);
      } else {
        console.log(ddata.toString());
      }
    });
  }
});
```

→ These Node.js packages provides essential functionalities to help developers build robust and feature rich applications.