Theory Assignment-1

Name:- Jay Tailor

Semester:- 7th

Roll no:- 74

subject : Full stack Development

code:- 701

☆ Node.js:- Introduction, Features, execution, architecture.

⇒ Introduction:- Node.js is an open source server enviroment. It is free.

- Node.js runs on various platform (windows, Linux, Unix, mac OS X, etc).
- Node.js uses Javascript on the server.

→ Node.js uses asynchronous programming.
- A common task for a web server can be to open a file on the server and return the content to the client.

→ Node.js can generate dynamic page content.
It can create, open, read, write, delete, and close files on the server.

⇒ It can collect form data.

→ Node.js can add, delete, modify data in your database

→ Node.js files contains tasks that will be executed on certain events. A typical event is someone trying to access a port on the server. Node.js files must be initiated on the server before having any effect. Node.js files have extension of ".js".

→ Features:- The following are the NodeJs features which makes it popular for developers.

- Asynchronous And Non-Blocking :- NodeJs is a Javascript environment which uses a single thread to handle all the request. All the request are handled synchronously, meaning the client side never needs to wait for data when requested. It also means it is non-blocking ; if a request is being processed in the CPU, the I/O operations are not blocked.

- written in Javascript:- Javascript is one of the most used programming languages currently in the industry. Almost all web applications have Javascript running. This makes it easy for web developers already familiar with Javascript to learn NodeJs quickly.

- Scalable:- Another of the NodeJs feature is scalability. This means that NodeJs can handle a large number of requests when needed. NodeJs can handle many requests due to its "single thread event-driven loop." It also has a cluster module that manages load balancing for all available CPUs.

- Cross-Platform Compatibility:- NodeJS features include cross-platform compatibility. This means it can run on many operating systems like mac, Linux and windows, to name a few. This helps NodeJS run efficiently on many devices without any worry about compatibility and provide all its features.

- Single-Threaded Event Driven Loop:- one of the NodeJS features is that it uses a single thread to handle all the requests. This thread is an event-driven loop; as each request is an event; once the event is processed, all the callbacks to that event are processed. This loop of handling requests and subsequent callback makes NodeJS a very efficient runtime when there are multiple lightweight requests

→ Execution architecture :- There are three different features like :-
① Event- driven architecture
② I/O model
③ Asynchronous architecture

→ ① Event driven architecture :- The Node JS is a single - threaded environment which means that it runs one action at a time and runs its operations smoothly with the use of events and event emitters.

→ ② I/O model :- The Input/output model is a concept used in Node runtime which accepts inputs that includes reading and writing files to disk, communicating with the servers, https request and security protocols among other concepts which are supported by libuv. The I/O method is used to coorm on data inputs and outputs with the use of callbacks, ~~referred~~ referred to as handlers, which inject data to these callbacks.

→ ③ Asynchronous architecture :- The Node JS runtime runs a single - threaded operation and to aid this operations, it runs an asynchronous non-blocking operation that allows it to carry on several operations at a time. This means while running one operation, it doesn't necessarily wait for this one to get the result of an operation and return it.

* Note on modules with examples.

→ In Node JS, it considers modules to be the same as Javascript libraries.
A set of functions you want to include in your application

→ Built in modules :-

- Node.JS has a set of built-in modules which you can use without any further installation.

- To include a module, use the 'require()' function with the name of the module:

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

- Now your application has access to the HTTP module, and is able to create a server:

```
http.createServer (function (req, res) {
    res.writeHead (200, {'content-Type':
                         'text/html'});
    res.end ('Hello World');
}).listen (8000);
```

- Create Your Own Modules:-
You can create your own modules, and easily include them in your application.

The following example creates a module that returns a date and time object:-

Example:-

Create a module that returns the current date and time

```
exports.myDateTime = function () {
    return Date();
};
```

→ Include your own module

Now you can include and use the module in any of your Node.js files.

Example:
```
var http = require('http');
var dt = require('./myFirstmodule');
http.createServer(function (req, res)
{
    res.writeHead(200, {'Content-Type':
                        'text/html'});
    res.write("The date and time are
              currently: " + dt.myDateTime());
    res.end();
}).listen(8000);
```

Note On package with example

→ A Package in Node.Js 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 package called "upper-case";

- Download "upper-class";
  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 package you install in the future will be placed in this folder.

- my Project now has a folder structure like this:

  C:\Users\my Name\ node-modules\ upper-case.

- Using a package:- once the package is installed, it is ready to use
- Include the "upper-case" package the same way you include any other module:

  Var uc = require('upper-case');

- create a node.js file that will convert the output "Hello World!" into upper-case letters.

Example:-

```
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(8000);
```

* use of package.json and package-lock.json

package.json is a metadata file in Node.js project that describes the project's dependencies, scripts, configuration, and other details.

- It typically contains information about the project such as its version, author, and license. It also lists the project's dependencies on other Node.js packages along with their version numbers, so that these dependencies can be automatically installed when the project is set up or updated.

- The package.json file is typically located at the root directory of a node.js project and is automatically generated when you run "npm init" command to initialized a new project. You can also manually create and modify this file to manage your project's dependencies and configuration

- This file is essential for Node.js projects as it provides a standardized way to manage dependencies and define project configuration, making it easier to share and collaborate on projects with others.

Example:- Two ways to generate
For using npm, run "npm init" on your terminal.
For using yarn, run "yarn init" on your terminal.
You can also run "npm init -y" or "yarn init -y" to generate the package.json.

Example:- {
"name": "package.json-file",
"version": "1.0.0",
"desc": "creating a package.json file",
"private": false
"main": "index.js",
"scripts": {
"start": "node index",
"dev": "nodemon index",
"test": "jest"
},
"repository": {
"type": "git",
"url": "git+http://github.com/
Easybuoy/Package-json
master.git" },
"homepage": "https://github.com/
Easybuoy/Package.json-master
#readme",
"engines": {
"npm": "6.10.0",
"node": "10.14.1"
},
}

→ The package.json file did not provide a way to lock down the specific version of each dependency that a project was using. This meant that when a project was deployed or shared with others, there was a risk that different developers or machines would use different versions of the same dependency, which ~~could~~ could cause compatibility issues or unexpected behaviours.

- package-lock.json file is like a ~~one~~ one-stop solution of your earlier problem. Package-lock.json is a file that is automatically generated by npm when a package is installed. It records the exact version of every installed dependency, including its sub-dependencies and their versions.

- The purpose of package-lock.json is to ensure that the same dependencies are installed consistently across different environments, such as development and production environments. It also helps to prevents issues with installing different package versions, which can lead to conflicts and errors.

- "package-lock.json" is created by npm when you run the npm install command. It contains a detailed list of all the packages, their dependencies, their specific version numbers, and locations.

Example:-

```
{
  "name":- "que-form",
  "version":- "1.0.0",
  "lockfileVersion": 1,
  "requires": true
  "dependencies": {
    "@babel/code-frame": {
      "version": "7.8.3",
      "resolved": "https://...",
      "dev": true
      "requires": {
        "@babel/highlighted": "^7.8.3"
      }
    }
  }
}
```

Node.js packages

- A package is a folder tree described by a package.json file. The package consists of the folder containing the package.json file and all subfolders until the next folder containing another package.json file, or a folder named node_modules.

- In package's package.json file, two fields can define entry points for a package: "main" and "exports". Both field apply to both ES modules and CommonJS module entry points.

- The 'main' field is supported in all versions of Node.js, but its capabilities are limited: it only defines the main entry point of the package.

- The "exports" provides a modern alternative to "main" allowing multiple entry points to be defined. conditional entry resolution support between environments and preventing any other entry points besides those defined in "exports".

- for new packages targeting the currently supported versions of Node.js, the "exports" field is recommended

- To make the introduction of "exports" non-breaking, ensure that every previously supported entry point is exported

```
{
  "name": "my-package",
  "exports": {
    ".": "./lib/index.js",
    "./lib": "./lib/index.js",
```

NPM introduction and Commands with its use

- Although you might see different variations of the meaning of NPM, the acronym stands for "Node package manager."

- npm is a package manager for node.js projects made available for Public use. Projects available on the NPM registry are called "packages"

- NPM allows us to use code written by other easily without the need to write ourselves during development.

- The npm registry has over 1.3 million packages being used by more than 11 million developers across the world.

- npm install

- This command is used to install Packages. You can either install Packages globally or locally. We it's install globally, it can make use of the Package's Functionality from any directory in our computer.

npm uninstall

- This command is used to uninstall a package

- npm init

- The init command is used to initialized a proj. When you run this command, it creates a package.json file

- npm update

- Use this command to update an npm package to its latest version

- npm restart

- Used to restart a package. You can use this command when you'd want to stop and restart a particular project

- npm start

- Used to start a package when required.

- npm stop

- Used to stop a package from running.

- npm version

- shows you the current npm version installed on your computer.

* Describe use and working of following Node.js packages.

→ Url: The url module splits up a web address into readable parts.
- To include the URL module, use the 'require()' method:

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

→ Process pm2- PM2 is a free open source, efficient, and cross-platform production-level process manager for node.js with a built-in load balancer.

install the PM-2
:- npm i -g pm2

→ Readline: The readline module is used in reading input and writing output. It provides a way to prompt different types of messages while giving the option to enter the input. To use this module, create a new JS file and write the following code at the start of the application.

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

→ - Fs :- Node.js file system module allows you to work with the file system on your computer.

To include the file system module, use the require() method:

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

→ Events :- Every action on a computer is an event. Like when a connection is made or a file opened.

- Objects in Node.js can fire events, like the readStream objects fires events when opening and closing a file:

```
var events = require('events');
var eventEmitter = new events.EventEmitter()
```

→ console :- Node.js console is a global object and is used to print different levels of messages to stdout and stderr.

```
Console.log ([data] [,...])
```

→ Buffer :- Node provides Buffer class which provides instances to store raw data similar to an array of integer but corresponds to a raw memory allocation.

```
var buf = new Buffer(10);
```

Query String :- The query string module used to provides utilities for parsing and formatting URL query string.

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

→ HTTP :- To make HTTP request in node.js there is a built-in module HTTP in node.js to transfer data over the HTTP.

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

→ V8 :- V8 is the name of javascript engine that powers google chrome. It's the thing that takes our javascript and execute it while browsing with chrome

→ **OS**:- OS is a node module used to provide information about the computers operating system.

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

→ **zlib**:- The zlib module provides a way of zip and unzip files.

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

* Important properties and methods and relevant programs.

→ URL :- The URL property setting is the target address OR the link. There are many methods to specify. Forexample, http://, https://, ftp://, gopher://, etc. It stands for uniform Resource Locator.

type://address/path.

→ - url.parse(urlstring[, parseQuerystring[, slashesDenoteHost[]]);

Parse a url string and returns an object ~~to~~ process a url string and returns an object with url component.

ex:- const urlstring = "https://www.excum.:8000/path"
const parseurl = url.parse(urlstring, true);
console.log(parseUrl);

Parse an address with url parse() method, it will return a url object with each part of the address as properties

Var q = url.parse(adr, true);

→ Process pm2 (external Resources):

Process.argv :- An array containing command line arguments.

Process.env :- An object containing the environment variables.

process.exit([code]) :- Exists the current process with an optional exit code.

~~exit~~ exec, pid, Platform, release, Version, version etc.

ex:-

```
Console.log (Process.argv);
Console.log (Process Architecture: ${process.exec});
Console.log (Process PID ${process.pid});
Console.log (Process Version : ${process.version});
```

Cwd() :- returns Path of current working directory.
hrtime() :- returns the current high-resolution real time in a [seconds, nano seconds]

→ Readline:

clearLine():- clears the current line of the specified stream.

clearScreenDown():- clears the specified stream from the current cursor down position.

createInterface(): creates an Interface object

cursorTo(): moves the cursor to the specified position.

moveCursor():- moves the cursor to a new position, relative to the current position.

Ex:-
```
var readline = require('readline');
var fs = require('fs');

var lineno = 0;
my Interface on('line', Function, Cline){
    lineno++;
    console.log('Line number' +
        lineno + ':' + Cline);
});
```

→ **fs:-** fs.readFile(): used to read files on your computer

fs.appendFile(): appends specified content to a file If the file dose not exits, the file will be created.

fs.open(): task takes a flag as the second argument, if the flag is 'w' for writing, the file is opened for writing.

Ex:-

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

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

const content 'This is write';
fs.writeFile('output.txt','content,
                (err) => {
          if (err) throw err;
          console.log('Data written');
        });
```

→ **events**:- All event properties & methods are an instance of an Event Emitter object. To be able to access these properties and methods.

Ex:- `var events = require('events');`
`var eventEmitter = new events.EventEmitter();`

→ **console**:-

```
console.log([data][,...]);
console.info([data][,...]);
console.error([data][,...]);
console.warn([data][,...]);
```

Ex:- `console.info("Program Started");`
`var counter = 10;`
`console.log("counter = %d", counter);`

* Buffer:-

→ Buffer:- alloc(), concat(), fromJ('abc');

console.log(Buf);

Create log an empty Buffer;

Var Buf = Buffer.alloc(15);

→ Query string:-

escape(): returns an escaped querystring

Parse(): Parses the querystring & return an Object.

string fy(): stringfies the object, returns a querystring.

→ http:- http.createServe([Options], [request listener])

create an HTTP instance