Assignment - 1

1. Nodejs : Introduction, features, execution architecture

→ Introduction :

- Node.js is an open-source, cross-platform Javascript run-time environment that executes Javascript code outside of a browser.
- Node.js was written and introduced by Ryan Dahl in 2009.

→ Features :

1. Extremely fast :

- Node.js is built in Google Chrome's V8 Java Script 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. The server moves to the next API after calling

it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.

3. Single threaded :

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

4. Highly Scalable :

- Node.js 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. Node.js applications never buffer any data. These applications 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.
License: Node.js is released under the MIT license.

→ Execution Architecture:

- The Execution Architecture of Node.js is based on the following components:

→ V8 Engine:

- Node.js uses the V8 Javascript engine, developed by Google, which compiles Javascript code into native machine code for faster execution.
- V8 provides high-performance runtime capabilities and allows Node.js to execute Java script code efficiently.

→ Event Loop:

- The event loop is at the core of Node.js and enables its non-blocking I/O operations.
- It works by handling and dispatching events to corresponding event handlers.
- This mechanism allows Node.js to execute multiple tasks concurrently without blocking the execution of other tasks.

→ Libuv :

- Libuv is a cross-platform library that provides the event loop implementation for Node.js.
- It abstracts the underlying operating system's I/O capabilities, allowing Node.js to be compatible with different platforms.

→ Callback and Event Handlers :

- Node.js follows an asynchronous programming model using callbacks & event handlers.
- Asynchronous functions are used to perform I/O operations, and once these operations complete, the corresponding callback or event handler is executed.

→ Node.js APIs :

- Node.js provides various built-in modules and APIs that allow developers to interact with the file system, network and other functionalities.
- These APIs are non-blocking, which enables efficient handling of concurrent tasks.

DATE
PAGE No

2. Note on modules with example.

→ 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.
- Node.js uses the commonJS module standard implementation in its module ecosystem.

→ Types of Modules:

- In Nodejs, there is 3 type of modules namely

- Core Modules
- Local Modules
- Third-party Modules.

1] Core Modules:

- Node.js 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 Java Script type depending on your module.

Syntax:

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

→ How to use Core Module:

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

Example:

index.js

```
const fs = require('fs');
fs.writeFileSync('noter.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

→ The List of Some Nodejs Core Modules :

→ fs - To handle the file system.
→ http - To make Node.js act as an HTTP server
→ path - To handle file paths
→ events - To handle events
→ url - To parse URL strings
→ querystring - To handle URL query strings.

2] Local Modules :

- Putting all your code in a single file is not a good idea.
- As you add 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 utils.js.

utils.js

```
const tests = function()
{
    console.log("Yehh! Local file is running.");
}
module.exports = tests;
```

→ Importing your own files:

- Now we need to import the local file into index.js. require function is also used for importing the files in the directory.

index.js

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

3] Third-party Modules:

- Third-party Modules can be installed from the NPM available online.
- Firstly we need to initialize the npm using the npm init command before npm can be used.

- It creates 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.

→ Installing an NPM module:

npm install "module-name"

- Example of Third-Party module:

- npm install express
- npm install lodash
- npm install mocha

→ validator is a popular module used for string validations and sanitizations.

- npm modules can be imported into your script using require. To load in an npm modules, pass the npm module name to require.

index.js

```
const validator = require("validator");
console.log("URL is" + validator.isURL
                ("https://google.com"));
console.log("Email is" + validator.isEmail
                ("johndoeeg.com"));
```

DATE
PAGE No

3. Note on package with example.

→ Node.js is a popular runtime environment for executing Java script code outside of a web browser.

- It comes with a package manager called npm, which allows developers to easily install and manage third-party packages to enhance the functionality of their Node.js applications.
- These packages are hosted on the npm registry and can be easily accessed and integrated into your Node.js projects.

→ Here's note on using packages with examples in Node.js :

i] Installing Packages :

- To use a package in your Node.js project, you need to install it first using npm.
- Open a terminal or command prompt, navigate to your project directory and run the following command to install the package :

```
bash
npm install package-name
```

- Replace 'package-name' with the actual name of the package you want to install.
- For example, to install the popular "lodash" package:

```
bash
npm install lodash
```

2] Using packages in Your code:

- After installing a package, you can require it in your Node.js code and start using its functionality. Here's an example of how to use the "lodash" package:

```
Javascript

// importing the 'lodash' package
const _ = require ('lodash');

// Using lodash functions
const numbers = [1, 2, 3, 4, 5];
const sum = _.sum(numbers);
console.log ('sum:', sum); // Op : sum: 15

const shuffledNumbers = _.shuffle(numbers);
console.log ('shuffled Numbers :', shuffleNumber);
// Output : Shuffled Numbers : [3, 2, 5, 1, 4]
```

DATE
PAGE No.

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

→ Package.json :

- Package.json is a JSON (Javascript Object Notation) file that contains metadata about the Node.js project, including its name, version, description, entry points, scripts and most importantly, its dependencies and devDependencies.
- This file serves as the manifest of the project and allows developers to define and manage the packages used in the project.

→ Here are some common uses of 'package.json'

1. Project Metadata
2. Dependencies
3. DevDependencies
4. Scripts
5. Version Management
6. Dependency locking

→ Example :

```
{
  "name" : "my-node-project",
  "version" : "1.0.0",
  "description" : "A simple Node.js project",
```

```
  "main": "index.js",
  "scripts": {
      "start": "node index.js"
  },

  "dependencies": {
      "express": "^4.17.1",
      "lodash": "^4.17.21"
  },

  "devDependencies": {
      "nodemon": "^2.0.12"
  }
}
```

→ Package-lock.json :

- Package-lock.json is a file automatically generated by npm (Node Package Manager) when you install or update packages in your project.
- It keeps a detailed record of the exact versions of all the installed packages, their dependencies and their sub-dependencies.
- This ensures that everyone working on the project installs precisely the same versions of the dependencies.

→ Here's how 'package-lock.json' is used:

1 Dependency Resolution
2 Deterministic Installs
3 Faster Installations
4 Security and Integrity
5 Continuous Integration and Deployment

→ Example:

```
{
  "name" : "my-node-project",
  "version" : "1.0.0",
  "lockfileversion" : 2,
  "requires" : true,
  "packages" : {
    "" : {
      "dependencies" : {
        "express" : "^4.17.1",
        "lodash" : "^4.17.21"
      }
    },
    "node_modules/express" : {
      "version" : "4.17.1",
      "resolved" : "https://registry.npmjs.org/...",
      "integrity" : "sha512-myTbmszEzpix ...",
      "dependencies" : {
      }
    },
  }
}
```

5. Node.js Packages

→ Node.js packages are collection of reusable code and functionalities that can be easily integrated into Node.js applications.

- Here are some types of Node.js packages:

1] Core modules :

- Node.js comes with a set of built-in modules that provide essential functionalities, such as file system operations, HTTP server creation, URL Handling and more.

2] NPM packages :

- These are packages available on the Node package manager registry.
- Developers can install them using the 'npm' command and use them in their projects.

3] Custom packages :

- Developers can create their own Node.js packages to encapsulate specific functionalities or components that

they want to reuse across multiple projects.

- These packages can be published on the npm registry or kept private within an organization.

→ Some popular Node.js packages and libraries include:

- Express :

A minimalist web application framework that simplifies building APIs and web applications in Node.js.

- Lodash :

A utility library that provides a collection of handy function for manipulating arrays, objects, strings and more.

- Socket.IO :

A library that enables real-time, bidirectional communication between web clients and servers using Websockets.

- Mongoose :

An object Data Modeling (ODM) library for MongoDB, providing a straight forward way to work with MongoDB databases in Node.js applications.

- Axios :

A widely used HTTP client that simplifies making HTTP request from Node.js.

- Joi :

A validation library that helps in defining and validating Javascript objects and data.

- Passport :

An authentication middleware that provides a flexible and modular way to handle user authentication in Node.js applications.

DATE
PAGE No

6. NPM introduction and commands with its use

→ NPM:

- npm is a package manager for Node.js, and it stands for "Node Package Manager".
- It is a command-line tool that allows developers to easily install, manage, and share packages of reusable code.
- These packages can include libraries, frameworks, tools and any other code that can be used to enhance your Node.js projects.

→ NPM command with their uses:

1) npm init

- Use: Initializes a new Node.js project and creates a 'package.json' file with basic project information
- Ex: 'npm init'

2] npm install

- Installs dependencies listed in the 'package.json' file into the 'node_module' folder
- Ex: 'npm install'

3] npm install package-name

- Installs a specific package and adds it to the 'dependencies' section in the 'package.json'
- Ex: 'npm install express'

4] npm uninstall package-name

- Uninstalls a package from the project and removes it from the 'dependencies' or 'devDependencies' section in the 'package.json'
- Ex: 'npm uninstall lodash'

5] npm update

- Updates all packages in the project to their latest versions.
- Ex: 'npm update'

6] npm outdated

- Displays a list of installed packages that are outdated, showing their current and latest versions.
- Ex: 'npm outdated'

7] npm list

- Lists all installed packages and their dependencies in a tree-like format.
- Ex: 'npm list'

8] npm run script-name

- Executes a script defined in the 'scripts' section of the 'package.json'
- Ex: "scripts": { "start": "node app.js" }

9] npm publish

- Publishes your package to the npm registry, making it publicly available for others to use
- Ex: 'npm publish'

10] npm link

- Creates a symlink from a globally installed package to the current project, enabling you to work on the package locally.
- Ex: 'npm link my-package'

7. Describe use and working of following Nodejs packages also important properties and relevant programs.

→ URL :

- The url module slips up a web address into redable parts

To include the url module, use require () method,

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

- parse an address with the url.parse() method, and it will return a url. project with each part of the address as properties

```
var url = require ('url');
var adr = "http://localhost:8080/default.
  html ? year = 2017 + month = 'february';

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

console.log (q.host);
console.log (q.pathname);
console.log ( q.search);
```

var detail = query;
console.log(qdetail, month);

→ Processe, pm2 (external package):

- pm2 (process manager 2) is a popular production-ready process manager for Node.js application. It helps you manage and deploy your node.js application ensuring high-availability reliablity and easy scaling.

- pm2 provides features like process monitoring automatic restarts, load balancing and more, making it a great tool for managing node.js app in production environment.

→ Process management :-

- pm2 can stop, start and restart process easily enguring that your Node.js application running continsly.

→ Local balancing :-

- pm2 allow you to run multiple instances of your application to distribute incoming request.

→ Automatic Restart :

- pm2 can automatically restart your application if is fresher or encounter errors.

→ readline

- Readline module in Node.js allows line by line this module wraps up the process standard output and process standard input objects.

- Node js file system allows you to work with the file system on your computers

var fs = require ('fs');

→ common use of file system module :

- Readfile
- createfile
- Updatefile
- Delete file
- Rename file

→ Read files :

- The fs.readfile() method is used to read files on your computer

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

http : crectesrver ( function (req, res)
{
  fs.readfile ('demo.html' function
                err, data)
  {
    res.write (data);
    return req.end ();
  });
}).listen (8000);
```

- create file :

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

fs.appenfile ("file.txt" Hello content
function (err)
{
    if (err) . throw err;
    console.log ('solved');
}
```

→ event :-

- To include the built-in event module use the require() method in addition all event properties and method are an instance of an eventmitter object.

```
var events = require ('event').EventEmitter();

var eventemitter = new event.EventEmitter();

var myEventHandler = function()
{
    console.log ("I have a screen");
}

eventemitter.on ('screen', myEventHandler);
event emitter.emit ('screen');
```

→ Console :

- Node.js console module object that provides a simple console similar to Java script to display different levels of message.

- Example :

```
const fs = require('fs');
const out = fs.create write stream
                ('/st out.log');
const err = fs.create write stream
                ('.stderr.log');
const myobject = new console.console
                (out, err);
myobject.log('This is the first example');
```

global console :- It is used without calling require('console');