# THEORY ASSIGNMENT - 1

NAME - KRISHI TAILOR

ROLL NO - 75

SEMESTER - 7th

SUBJECT - APPLICATION DEVELOPMENT USING FULL STACK

Q.1 Node js - Introduction, features, execution architecture.

Ans 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 application. It is open-source and free to use. Many of the basic modules of nodejs are written in javascript. Node js is mostly used to run real time server applications. Node js also provides a rich library of various javascript module to simplify the development of web applications.

Node.js = Runtime Environment + Javascript library

Features of Node js :-

* Extremely fast :- Node js is built on google chrome's V8 javascript engine, so its library is very fast in code execution.

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

* Highly Scalable :- Node js is highly scalable because mechanism helps the server to respond in non-blocking way.

* License :- Node js is released under the MIT license.

* Open Source :- Node js has an open source community which has produced many excellent modules to add additional capabilities to node js application.

* No buffering :- Node js cuts down the overall processing time while uploading audio and video files. Node js doesn't buffer any data.

* I/O is Asynchronous :- All API of node js library are asynchronous i.e, non-blocking. So, node js based server never waits for an API to return data.

Execution Architecture :-

* Event Loop :- The event loop is responsible for handling asynchronous operation and I/O task efficiently. It keeps an eye on the message queue and executes the call back functions associated with completed I/O operations when the call stack is empty.

* Single threaded, Non-Blocking :- Node js operates on a single threaded event loop. This might seem counterintuitive for handling multiple concurrent requests; but node js achieves this through non blocking I/O operations. When a request requires an I/O operation, node js delegates the task to the system.

* Callbacks and Asynchronous Programming :- Callbacks play a significant role in node js instead of waiting for a function to complete before moving to the next one, node js uses callbacks to notify the completion of an asynchronous task.

* Event Driven Architecture :- Node.js utilizes an event driven architecture to handle events and execute associated callbacks when an event occurs. Event can be anything from I/O operation to Http requests. The event loop continous listens for events.

* Libuv :- Node.js relies on a library called libuv to handles its asynchronous I/O operations. Libuv abstracts the underlying system calls and provides a consistant interface accross different platforms. It manages threads, handles file I/O networking and timers efficiently to optimize the event loop's performance.

Q.2 Noti on modules with example.

Ans In node js, modules are reusable blocks of code that encapsulate functionality and can be imported and used in other parts of the application. Modules help in organizing code, promoting code reusability and maintaining a clean codebase.

There are different types of modules:-

1.) Built in core modules :-
Node.js comes with several core modules that provides essential functionalities. These modules can be used directly without the need for any additional installation.

Example :-

```
const fs = require ('fs');
fs.readFile ('example.txt', 'utf-8', (err, data) => {
    if (err) {
        console.error("Error reading file");
    }
    else {
        console.error ("File contents");
    }
});
```

2.) Local modules :-
Local modules are custom modules created by developers within their project. These modules can be imported into other files using relative paths.

Example:-

```
module.exports = {
    calculateArea: function (width, height) {
        return width * height;
    }
};
```

module in another file -

```
const rectangleModule = require('./rectangle');
const width = 5;
const height = 10;
const area = rectangleModule.calculateArea(width, height);
console.log("Area of rectangle", area);
```

3) Third party modules :-

Node.js has a vast ecosystem of third party modules available on npm registry. Developers can install these modules and use them in their projects.

Example:-

```
const axios = require('axios');
axios.get('https://api.example.com/data')
  .then((response) => {
      console.log("Data Recived", response.data);
  })
  .catch((error) => {
      console.log("Error", error.message);
  });
```

Q3 Note on packages with example

Ans Packages are collections of reusable code that can be easily imported into your projects to add specific functionality. They help in organizing and structuring your code, making it more maintainable and efficient. Node.js uses the CommonJS module system to manage packages, which enables developers to easily require and use modules in their applications.

Using packages in node js :-

1. Installing packages :-
To use a package in your Node.js project, you need to install it first. Node.js comes with the npm (Node package manager) that allows you to install packages from the npm registry.

To install a package, open your terminal and run :-

```
npm install <package-name>
```

2. Requiring packages :-
Once the package is installed, you can require it in your node.js code using the 'require' function.

Example :-

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

3. Using the package :-
After requiring the package, you can use its functions by accessing its methods and properties.

Example :-

```
const axios = require('axios');
axios.get('http://jsonplaceholder.typecode.com/1')
    .then(response => {
        console.log(response.data);
    })
    .catch(error => {
        console.log('Error', error);
    });
```

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

Ans

1. Package.json :-

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

→ It includes information like project name, version, author, license and most importantly, the list of dependies required for project to run.

→ When you create a new Node.js project or run 'npm init', it prompts you for information to generate the initial 'package.json' file.

→ You can also manually edit 'package.json' to add or remove dependencies, scripts and other project specific configurations.

→ Example :-

```
{
  "name" : "my-node-app",
  "version" : "1.0.0",
  "discription" : "sample",
  "main" : "index.js",
  "dependencies" :
    {
      "express" : "^4.17.1",
      "axios" : "^0.21.1"
    },
```

```
"scripts":
{
    "start": "node index.js"
},
"author": "Name",
"license": "MIT"
}
```

2. Package-lock.json :-

→ The 'package-lock.json' file is automatically generated by npm and is used to lock the version of each installed dependency.

→ It ensures that all installations of the project use the exact same versions of the dependencies, making the project's behaviour consistant across different environments.

→ This file is automatically updated by npm whenever you install, update, or remove packages using 'npm install' or other commands.

→ The 'package-lock.json' file is crucial for projects that are meant to be shared or deployed on different machines, as it ensures that everyone working on the projects is using the same versions of the dependencies.

→ Example:- {
"name" : "my-node-app",
"version" : "1.0.0",
"lockfileVersion" : 2,
"requires" : true,
"dependencies" : {
"axios" : {
"version" : "0.21.1",
"resolved" : "https://registry.npmjs.org/axios/-/axios-0.21.1.tgz",
"integrity" : "sha512-----"
},
"express" : {
"version" : "4.17.1",
"resolved" : "https://registry.npmjs.org/express/-/express-4.17.1.tgz",
"integrity" : "sha512-----"
}
}
}

Q.5 Node.js packages.

Ans In the Node.js ecosystem, packages are collection of modules, libraries and resources that developers can use to enhance their applications.

1. Web frameworks :- Packages like express.js are popular web frameworks that simplify the process of building web applications and APIs by providing routing, middleware support and other features.

2. Utility libraries :- Packages like lodash, Ramda and underscore.js provide utility functions that assist with tasks like data manipulation, validation and functional programming.

3. Database libraries :- Packages like Mongoose and sequelize provide easy-to-use abstractions for working with databases and ORM (Object relation mapping) capabilities.

4. Authentication and Security :- Packages like passport.js offer solutions for authentication and password hashing to enhance application security.

5. Template engines :- Packages like EJS, Pug and handlebars enable developers to generate dynamic HTML content easily.

6. HTTP clients :- Packages like Axios and request provide tools for making HTTP requests, allowing Node.js applications to interact with APIs and web services.

7. CLI (command line) tools :- Packages like commander and yargs enable developers to build interactive and user-friendly command line tools.

8. Real-time communication :- Packages like socket.IO facilitate real-time communication between clients and server using web sockets.

9. File system utilities :- Packages like fs-extra and glob provide additional functionalities and ease of use for working with file system.

10. Date/Time manipulation :- Packages like moment.js and day.js offer tools for parsing, formatting and manipulating dates and times.

11. Logging :- Packages like winston and Bunyan provide flexible logging solutions for node.js applications.

Q.6 npm introduction and commands with its use.

Ans. npm which stands for Node packages manager, is a command-line tool and the default package manager for node.js. It simplifies the process of installing, updating, and managing node.js packages and their dependencies. npm also provide a central repository where developers can publish and share their node.js packages, making it easy for others to use and contribute to their code.

npm commands with its use:-

1. Initializing a new node js project:-
To create a new nodejs project, you can use the 'npm init' command. It will guide you through a series of prompts to create a 'package.json' file, which contains information about the project.
   ⇝ npm init

2. Installing packages:-
To install packages for your node js project, use the 'npm install' command followed by the package(s) name.
   ⇝ npm install <package-name>
   npm install <package-name1> <package-name2>

Eg npm install nodemon
   npm install express

3. Uninstalling packages:-
To remove a package from your project, use the 'npm uninstall' command followed by package name.
⇝ npm uninstall <package-name>

Eg npm uninstall express

4. Using packages in your code:-
After installing a package, you can import and use it in your node.js code using the 'require' statement.

```
Eg const express = require('express');
   const app = express();
   app.listen(3000, () => {
       console.log('Server running on
           port 3000');
   });
```

5. Listing installed packages:-
To view a list of all installed packages, use the 'npm list' command.
⇝ npm list

6. Updating packages:-
To update package to their latest versions, use the 'npm update' command. To update a specific package, append the package name after the command.
⇝ npm update
npm update <package-name>

7. Running Scripts :-
You can define custom scripts in the 'scripts' section of your 'package.json' file. These scripts can be executed using the 'npm run' command.

Eg "scripts' : {
"start" : " node index.js"
}

8. Search for packages :-
To search for packages in npm registry, use the 'npm search' command followed by a search term.

⇝ npm search <search-term>

Eg npm search lodash

Q.7 Describe use and working of following nodejs packages. Important properties and methods and relevant programs.

Ans

1. url :-

→ The 'URL' module provides utilities for URL resolution and parsing. It is used to work with URLs and extract information from them.

Parsing a URL :-

```
const {URL} = require ('url');
const urlString = 'http://www.demo.com.8080/path?
                    query=hello--';
const parsedUrl = new URL (urlString);
console.log (parsedUrl.host);
console.log (parsedUrl.pathname);
console.log (parsedUrl.searchParams.get('query'));
```

2. process, pm2 (external package) :-

→ 'process' object provides info, & control over the node.js process. It allows interacting with current process and accessing environment variable.

Getting command-line arguments

```
console.log (process.argv);
```

→ 'pm2' is an external package used to manage node.js processes. It provides tools for process monitoring, scaling and cluster management.

↳ install pm2 globally

```
npm install -g pm2
pm2 start app.js
```

3. readline :-

→ The 'readline' module provides an interface for reading input streams line by line. It is commonly used to interact with users in the command line environment.

Reading user input :-

```
const readline = require ('readline');
const rl = readLine.createInterface ( {
        input : process.stdin ;
        output : process.stdout ;
} );
rl.question ('what's your name? ', (name) => {
        console.log ('Hello, ${name}!');
        rl.close ();
});
```

4. fs :-

→ The 'fs' module provides file system-related functionality, allowing reading, writing and manipulating files.

Reading a file :-

```
const fs = require ('fs');
fs.readfile ('example.txt', 'utf-8', (err, data) => {
    if (err) {
        console.error ("Error reading file', err);
    } return;

    console.log ("File Content: ', data);
});
```

5. events :-

→ The 'events' modules provides an event-driven architecture for building applications that can emit and listen to events.

```
const EventEmitter = require ('events');
class MyEmitter extends EventEmitter {};
const myEmitter = new myEmitter ();
myEmitter.on ('greet', (name) => {
        console.log ('Hello, ${name}');
});
myEmitter.emit ('greet', 'Me');
```

6. console :-

→ The 'console' module provides a simple debugging console that can be used to log messages during development.

```
console.log ('This is a log message');
console.error ('This is an error message');
console.warn ('This is waring message');
```

7. buffer :-

→ The 'buffer' module provides a way to handle binary data. It is used to work with raw binary data in node.js applications.

```
var buf = buffer.from ('abc');
console.log (buf);
```

8. querystring :-

→ The 'querystring' module provides utilities for working with query string in URLs.

```
const querystring = require ('querystring');
const params = querystring.parse ('name = abc & age = 10
console.log (params);
```

9. http :-

→ The 'http' module provides a set of functions and classes to create HTTP servers and make HTTP requests.

```
const http = require ('http');
const server = http.createServer ( req, res ) => {
    res.writeHead (200, {' Content-Type' : 'text' });
        res.end ('Hello', 'World');
});
Server.listen (3000, () => {
    console.log ('Server is running on port 3000'
});
```

10. V8 :-

→ The 'V8' module exposes APIs related to the V8 javascript engine, providing access to performance and memory related data.

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

11. OS :-

→ The 'os' module provides operating system related functionality such as information about the host operating system.

```
const OS = require ('os');
console.log('os platform', os.platform());
console.log('CPU architecture', os.arch());
```

12. zlib :-

→ The 'zlib' module provides compression and decompression functionalities using gzip and deflate.

→ Compressing and decompressing

```
const zlib = require ('zlib');
const data = "This is some data to compress";
zlib.gzip (data, (err, compressed data) => {
    if (err) {
        console.error ("compression error", err);
        return;
    }
    console.log ("compressed data", compressedData);

    zlib.unzip (compressedData, (err, decompressed
                                 Data) => {
        if(err) {
            console.error ("Decompressed error");
            return;
        }
```

```
        console.log('Decompressed data');
    });
});
```