

Name: Lakhuni Vivek S

Roll No: 25 Div: A

Semester: 7<sup>th</sup>

Subject: FOI Application  
Development  
using

Assignment: I Fullstack

Date: 18<sup>th</sup> July 23

Q 1. Nodejs : Introduction, Features, execution architecture.

Ans.

Nodejs 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 applications.
- It is open source and free to use.

### Features of Nodejs

1. Extremely fast:

Node.js is built on Google Chrome's V8 Javascript 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.

### 3 Single threaded:

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

### 4 Highly Scalable:

Node.js is 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.

### 7 License:

Node.js is released under the MIT license.

- A typical web application consists of the following components:

### Client :

A client refers to the user who interacts with the server by sending out requests.

### Server :

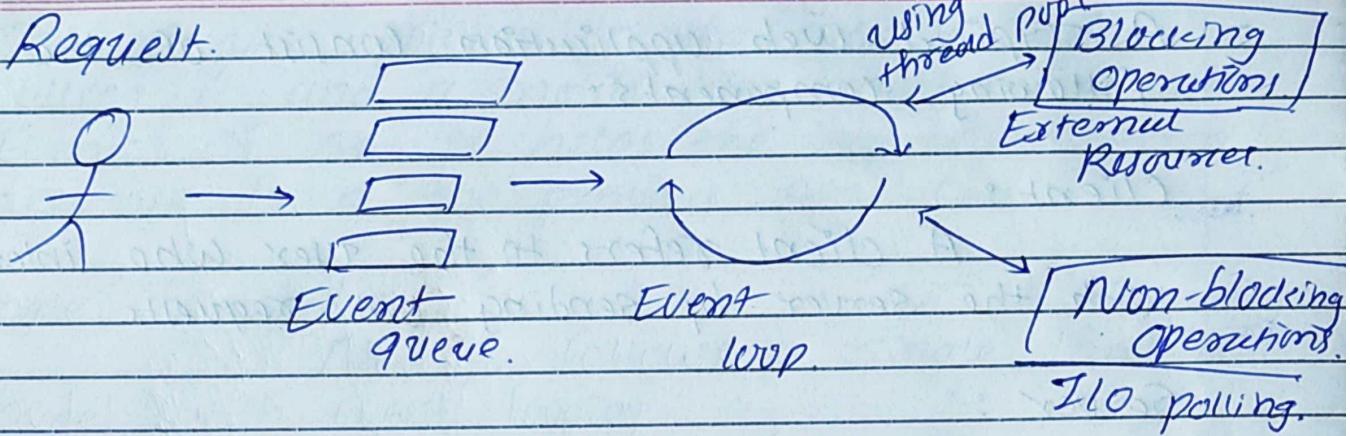
The server is in charge of receiving client requests, performing appropriate tasks, and returning results to the clients.

- It serves as a bridge between the front-end and the stored data, allowing clients to perform operations on the data.

### Database :

A database is where a web application data is stored. Depending on the client's request, the data can be created, modified and deleted.

- To manage several concurrent clients, Node.js employs a "single threaded event loop" design. The Javascript event-based model and the Javascript callback mechanism are employed in the Node.js processing model.
- It employs 2 fundamental concepts:
  1. Asynchronous model
  2. Non-blocking of I/O operations.
- Components of the architecture:
  1. Request
  2. Node.js Server
  3. Event Queue
  4. Thread Pool
  5. Event loop
  6. External Resources



Q. 2 Note on modules with example.

Ans. 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.
- Almost all programmers prefer modules because of their reusability throughout the application and ability to reduce the complexity of code into smaller pieces.
- Node.js uses the commonjs module standard implementation in its module ecosystem.
- There are 3 types of modules in node.js.
  1. Core modules
  2. Local modules
  3. Third-party modules.

I. 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 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 (require()) function will return a Javascript type depending on your module.

Syntax:

```
const module = require ("module-name");
```

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

Example :

```
const fs = require ('fs'); // Import fs module.
fs.writeFileSync ('abc.txt', 'I Love Nodejs');
```

→ core modules name :

fs	To handle the file system.
http	to make Node.js act as an HTTP server
https	To make Node.js Act As An HTTPS Server
os	It Provides Information about the operating system.
path	To handle file path.
cluster	To split a single node process into multiple process
dns	To do DNS lookups and name resolution functions.
tls	To implement TLS and SSL protocols
querystring	To handle URL query strings
url	To parse URI strings.
events	To handles events.
timers	To execute a function after a given number of ms.

## 2. Local modules:

Putting all your code in a single file 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.

### Exporting from files:

First of all, we need to create a file called myfunc.js file.

Now we can export JavaScript code written on this file using module.exports.

myfunc.js file.

```
const func = function() {  
    console.log("Hello, this is module");  
}  
module.exports = func;
```

### Importing our files:

Moreover, we need to import the local file into index.js.

index.js file

```
const myfunc = require('./myfunc.js');  
myfunc();
```

### Output:

Hello, this is module.

### 3. Third-party modules:

third-party modules can be installed from the NPM (Node package manager) available online.

- To use it, firstly we need to initialize the npm using npm init command and it will create package.json file in the root directory.
- It stores the all information about the third-party modules that we have installed as a dependency.

### Installing an NPM module:

npm install "module name"

Ex.   
= npm install express

npm install Validator

Q. 3. Note on package with example.

Ans. A package in Node.js contains all the files you need for a module.

- Modules are Javascript libraries you can include in your project.
- Downloading a package is very easy task.
- For that you need to open Command-line interface and tell NPM to download the package you want.

Ex.

=  
npm install 'package-Name'.

- NPM creates a folder named 'node\_modules', where the package will be placed. All the packages you install in the future will be placed in this folder.
- Once the package is installed, it is ready to use.
- Include the node-static package

```
const static = require('node-static');
```

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

```
var static = require('node-static');
```

```
const fileserver = static.Server('-1785');
```

```
http.createServer((req, res) => {
```

```
    res.writeHead(200, {'Content-Type': 'text/html'});
```

```
    res.fileServer.sendFile('ABC.html', res, {}), req, res);
```

```
    res.end();
```

```
}).listen(8080);
```

Dependence: The dependencies and dependencies for the package to work correctly.

## Use of package.json:

License: The license under which the package is distributed or project is.

Scripts: Custom scripts that can be run using development.

Dependencies: A list of packages that the project depends on at runtime.

Description: A brief description of the package.

Information including:  
- The contents essential metadata and configuration for a node.js project or package.  
Object notation) file that serves as a manifest for a node.js project.

package.json: The package.json file is a json (JavaScript object notation) file that serves as a manifest for a node.js project. They play a crucial role in managing dependencies and configurations for a package. This file contains different numbers and words defining constraints and behaviors of a package. They play a crucial role in managing dependencies and configurations for a package. This file contains different numbers and words defining constraints and behaviors of a package.

In Node.js, package.json and package-lock.json are fundamental files used to manage dependencies and configurations for a project.

Ans

Script execution: The script section allows you to define custom scripts that can be executed using 'npm run <script name>'.

- This makes it easier to manage common tasks associated with the project.

Version control: The package.json file is typically included in version control. This ensures that the package's configuration and dependencies are consistent across different development environments and when collaborating with other developers.

### package-lock.json:

The package-lock.json file is another json file that npm automatically generates when you run 'npm install'.

### Use of package-lock.json:

#### Dependency Version locking:

The primary purpose of package-lock.json is to lock the version of each installed dependency.

- It records the specific versions of each package and its transitive dependencies that were installed when 'npm install' was executed.
- This guarantees that anyone else who installs the project using the same package-lock.json will get exactly the same versions of all dependencies, ensuring consistency across different environments.

Faster and safer installs: By using it, npm can avoid resolving dependencies and their versions every time 'npm install' is run. Instead, it can quickly install the exact versions specified in the package-lock.json which makes installation faster and more reliable.

Security; it also helps enhance security by preventing potential security risks that might arise from using different versions of dependencies.

- since the exact versions are locked, any security patches applied to those versions will be automatically used when the project is installed

## Q.5. NPM introduction and commands with me.

Ans. NPM, which stands for node package manager, is a package manager for the Javascript programming language.

It comes bundled with Node.js, a runtime environment for executing Javascript code server-side.

NPM is widely used in the web development community to install, manage, and share open-source Javascript libraries and tools.

Here's a brief intro to npm and some commonly used commands:

### 1. Initialize a new project:

`npm init`

To start a new project, you first need to initialize a 'package.json' file.

- This command will prompt you to answer some questions to set up your 'package.json' file, or you can use 'npm init -y' to accept all default.

### 2. Install packages:

You can use npm to install packages from the npm registry and add them to our project's dependencies.

`npm install express`

- This will download the express package and add it to the dependencies section of our package.json file.

### 3. Install packages as development dependencies.

Sometimes we need packages only while the development process, such as testing libraries or build tools.

For that we use -- save-dev.

```
npm install package --save-dev.
```

### 4. Install packages globally:

Some packages provide command-line utilities that you want to use across different projects.

So we can install them globally -g.

```
npm install -g nodemon.
```

### 5. Update package:

To update package to latest or updated version to our project we can utilize of update

```
npm update.
```

### 6. Uninstalled packages

To remove a package from our project we use uninstall command.

```
npm uninstall node-static
```

### 7. List installed packages

To see a list of packages installed in our project we use ls command.

```
npm ls
```

### 8. NPM Run script:

In package.json file we can define custom script that we can run using npm run.

```
npm run start.
```

9. Search for packages:

We can search for packages in the npm registry.

npm search package-name.

10. publish your package:

If we've created a reusable library or tool, we can publish it to the npm registry for others to use:

npm publish.

Q. 6. Describe use and working of following node.js packages, important properties and methods and relevant programs.

Ans.

I. URL

- The URL module is built-in Node.js module that provides utilities for working with URLs.
- It allows us to parse, format and manipulate URLs.

Methods :

URL.parse ( URLString, parseQueryString [, ] );

- This method parses a URL string and returns an object containing its various components

Ex.

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

```
const urlString = 'https://www.google.com';
```

```
const parseURL = url.parse(urlString, true);
```

```
console.log(parseURL);
```

Output:

```
{
```

```
protocol: 'https:',
```

```
slashes: true,
```

```
auth: null,
```

```
host: 'www.google.com'
```

```
port: null,
```

```
},
```

### `URL.prototype.format(urlObject);`

This method takes a URL object and converts it back into a formatted URL string.

```
const URL = require('url');
const obj = {
  protocol: 'https',
  host: 'www.example.com',
  pathname: '/path/to/resource',
  query: { key: 'value' }
};

const formatedURL = URL.format(obj);
console.log(formatedURL);
```

Output:

```
https://www.example.com/path/to/resource?key=value
```

### `URL.resolve(from, to)`:

This method resolves a relative URL to an absolute URL.

Properties:

Protocol: 'http:', 'https:', etc.

Slashes: //

auth: 'username:password'

host: 'www.google.com'

PORT: port of URL

query: an object representation of the parsed query string.

## 2. Process:

The process is built-in global object in Node.js that provides information and control over the current node.js process.

- It allows you to interact with the process running your node.js application, access environment variables and handle events related to the process.

`process.argv`:

This property contains an array of command-line arguments passed to the Node.js application.

`node app.js arg1 arg2`

`console.log(process.argv);`

`process.env`:

This property contains an object representing the user environment.

`process.env.NODE_ENV`

`process.pid`:

This property holds the process id of the current node.js process.

`process.cwd()`:

This method returns the current working directory.

`pm2`:

pm2 is a process manager for node.js applications.

Starting an application: `pm2 start app.js`

Listing Running application: `pm2 list`

Stopping an application: `pm2 stop app-name`

Restart an application: `pm2 restart app-name`

View logs: `pm2 logs app-name`

## readline :

The readline module in node.js provides an interface for reading input from a readable stream in an interactive manner.

### Properties:

readline.createInterface(options):

This method creates a new Interface instance.

- It takes an options object as an argument, which specifies the input and output stream to use.

### Methods:

rl.question(query, callback):

This method prompts the user with the 'query' and waits for user input.

```
rl.question('what is your name?', (name) => {
```

```
    console.log(`Hello, ${name}!`);
```

```
});
```

rl.setPrompt(prompt):

This method sets the prompt that is shown to the user.

```
rl.setPrompt('enter name:');
```

```
rl.prompt();
```

rl.pause();

This method pauses the readline interface.

```
rl.pause();
```

rl.Resume();

This method resumes the readline interface.

fs;

The fs module in node.js stands for file system. It provides an interface for working with the file system, allowing you to read, write and manipulate files and directories on your computer or server.

Read file :

fs.readFile (path[, options], callback);  
Reads the content of the file.

```
const f1 = require('fs');
f1.readFile('ABC.txt', 'UTF8', (err, data) => {
    if (err) throw err;
    console.log(data);
});
```

fs.readFileSync (path[, options]);  
Reads the content of a file synchronously.

```
c.i.g.(fs.readFileSync('ABC.txt', 'UTF8'));
```

Write File :

fs.writeFile (file, data[, options], callback);  
Writes data to a file asynchronously. If the file does not exist, it creates a new one.

```
fs.writeFile('ABC.txt', 'Hello world',
            (err) => {if (err) throw err;
            console.log('File written successfully.');
});
```

## Events:

In node.js, the 'events' module is a core module that provides a mechanism to handle and emit events.

## EventEmitter class:

It is a core of a events module, it provides methods to handle and emit events.

## Event handling:

This method is used to register an event listener for a specific event.

```
myEmitter.on('event', () => {  
    console.log('event emitted');  
});
```

## Ex.

```
const EventEmitter = require('events');
```

```
class MyClass extends EventEmitter {
```

```
    constructor() {  
        super();  
    }
```

```
const obj = new MyClass();
```

```
obj.on('event', () => {  
    console.log('custom event');  
});  
obj.emit('event');
```

## Console

- The console is a built-in object in node.js and web browsers that provides a simple debugging and logging mechanism.
- It allows you to interact with the standard output and standard error streams to display messages, debug information and errors during the execution of your code.

Methods:

console.log():

Prints the provided data and objects to the console, separated by spaces.

- It automatically converts the values to strings.

```
const name = 'Vivek';
```

```
const rollno = '25';
```

```
console.log("name", name, "roll no",
```

```
rollno);
```

console.error():

Prints to the standard error stream instead of standard output.

```
console.error("this is error");
```

console.warn():

Prints to standard error stream, but it is often used for less severe warning or alerts.

```
console.warn('warning : low risk');
```

console.clear():

clears the console.

console.clear():

console.table();

Displays the data in tabular form.

const data = [

```
{ name: 'Vihok', rollno: 1 },  
{ name: 'Vihur', rollno: 2 }  
];
```

console.table(data);

console.assert();

Tests a value and prints an error message if the assertion fails.

const x = 10;

```
console.assert(x > 20, 'x should be greater  
than 20');
```

Buffer:

The Buffer class is a built-in global object that provides a way to handle binary data.

A Buffer is like an array of integers, but each element represents a single byte of data.

Creating "Buffer":

Buffer.alloc(5);

Buffer.from([65, 66, 67]); // create buffer.

Common Use Cases:

- Reading from and writing to streams, files and sockets

- Handling binary data in networking, protocols, encryption, and compression

- parsing and serializing binary data formats like BSON or Protocol Buffers

## QueryString :

The queryString is a part of the URL that comes after the question mark '?'. It is used to send data to the server as key-value pairs.

## Parsing Query strings:

```
querystring.parse([str[, sep[, eq[, options]]]);
```

## Creating query string:

```
querystring.stringify([obj[, sep[, eq[, options]]]);
```

- The querystring module in nodejs is helpful when dealing with URLs and processing query parameters in web applications and APIs.

## http :

The http module in nodejs is a core module that provides functionality to create HTTP servers and make HTTP requests.

- It allows you to handle incoming HTTP requests and serve responses, making it a fundamental building block for building web applications and APIs in nodejs.

## Creating an HTTP Server:

```
const http = require('http');
const Server = http.createServer((req,res) =>
{
    res.end('Hello World');
}).listen(8000);
```

V8:

V8 is an open-source JS engine developed by Google. It is written in C++ and is primarily used to execute JavaScript code in web browsers and server-side environments.

- V8 is at the core of many popular JS environments including Google Chrome, Node.js and several other runtime environments.

Features:

#### 1. Just-in-time (JIT) compilation:

V8 uses a Just-in-time compilation technique to optimize the execution of JS code.

- When a JS program is run, V8 first interprets the code and then compiles parts of it into highly optimized machine code for better performance.

This adaptive compilation helps improve the speed of executing JS.

#### 2. Garbage Collection:

V8 implements a garbage collector to manage memory allocation and deallocation for JS objects.

#### 3. Multi-platform support:

V8 is designed to be portable and can run on various platforms, including Windows, macOS, Linux and mobile operating systems like Android and iOS.

#### 4. Support for ECMAScript standards:

V8 is compliant with the ECMAScript Standard, which defines the syntax and semantics of the JS language.

## 5. Isolation and sandboxing:

V8 provides an isolation mechanism that allows running multiple JS contexts independently within the same process.

## 6. Profiling and Debugging:

V8 includes tools for profiling and debugging JS code. Developers can use tools like Chrome DevTools and Node.js Inspector to inspect memory usage, CPU performance and debug JS code.

## 7. Embedding in custom applications:

V8 is designed to be embedded, which means it can be integrated into custom application to allow those applications to execute JS code.

### (OS):

The 'os' module in node.js is a core module that provides operating system related utilities.

#### os.platform() :

```
const os = require('os');
console.log(os.platform());
// win32 output.
```

### zlib:

zlib module in node.js is a core module that provides compression and decompression functionalities using the Zlib library.

#### Compression:

```
zlib.deflate(buf[, options], callback)
```

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

```
const input = Buffer.from('Hello, zlib compression');
```

```
zlib.deflate(input, (err, compressed) => {
```

```
  if (err) { c.l.g.(err); }
```

```
});
```

## 2. zlib.inflateSync(buf[, options])

It decompressed the data.

```
const zlib = require('zlib');
const compressed = ... // Compressed data
const decompressed = zlib.inflateSync(
    compressed);
console.log('Decompressed data', decompressed);
```

## 3. zlib.gunzip(buf[, options], callback):

Decompresses the provided 'buf' using the gzip format.

```
const zlib = require('zlib');
const compressed = ... //
zlib.gunzip(compressed, (err, decompressed) =>
{
    if (err) {
        console.log('Decompressed data',
            decompressed.toString());
    }
});
```

## 4. zlib.gunzipSync(buf[, options])

Synchronous Version of zlib.gunzip()

- It decompressed the buf and returns the decompressed data.

```
const zlib = require('zlib');
const compressed = ...
const decompressed = zlib.gunzipSync(
    compressed);
console.log('Decompressed data =',
    decompressed.toString());
```

- The 'zlib' module in node.js is a powerful tool for compressing and decompressing data using various algorithms and formats.