* Nodejs

=) Nodejs is server side platform built on google chrome's Javascript engine (V8 engine).

=) It develop by Ryan Dahl in 2009

=) Latest version is v0.10.36.

=) Node Js is an open source, cross platform runtime enviroment for developing server side and networking applications.

=) Nodejs application are written in Javascript and can be run within the node.js runtime on OS X, microsoft window and linux.

=) Node Js also provide a rich library of Java script module which simplifies the development of web application using Nodejs to a great extent.

Node.js = Runtime environment + Js library

* Features

1. Asynchronous and Event Driven :-

All API of Nodejs library are asynchronous, that is non blocking. it essentially mean a node.js based server never wait for an API to return data. Server move to next API after calling it and a Notification machanism of Event of Nodejs helps the server to get a Response From the previous API call.

2. Very Fast:-

Built in google chrome V8 Javascript Engine, Node.js library very fast in code execution.

3. Single Threaded but Highly scalable:-

Nodejs uses single threaded model with event looping. Event mechanism helps the server to response in a non-blocking way and making the server highly scalable as opposed to traditional server which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of request than traditional server like Apache HTTP server.

4. No Buffering:-

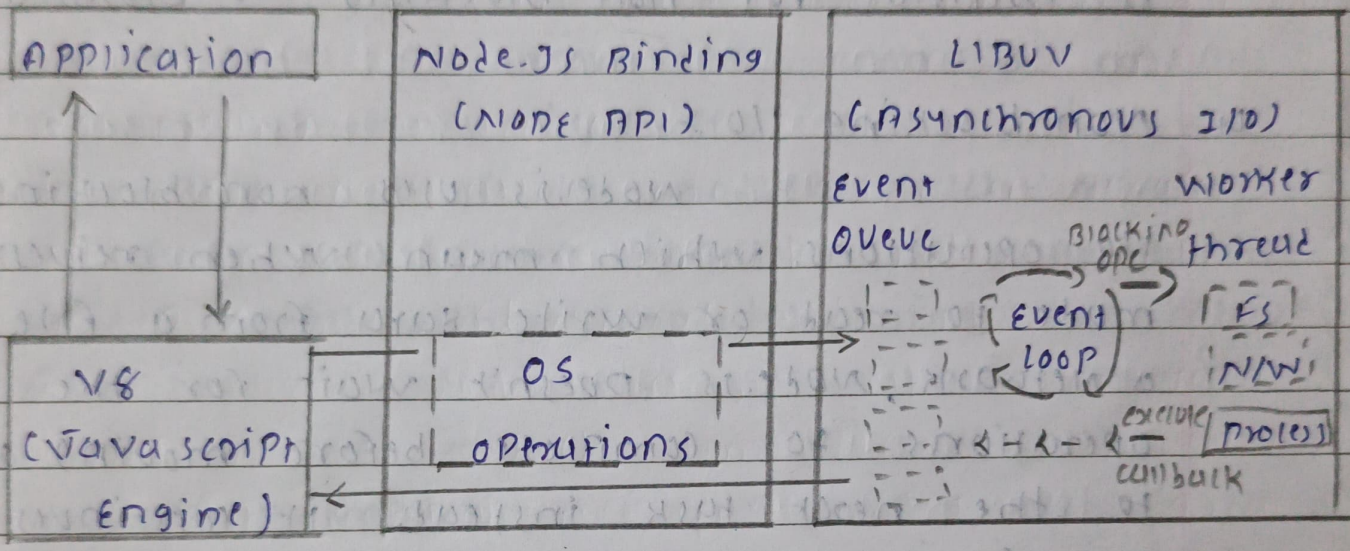
Node.js application never buffer any data. these application simply output the data in chunks.

5. License:-

Nodejs is released under the MIT License.

* Node.js execution architecture

=> Node.js follows a single-threaded, event driven and non-blocking I/O model, which allow it to handle a large number of concurrent connections efficiently.



=) 1. V8 Engine :-

Node.js relies on the V8 engine to interprit and execute JavaScript code. V8 is written in C++ and is used by google chrome as well. it compile JavaScript code into machine code using just-in-time compilation, optimizing the code for performance.

2. Single threaded event loop :-

This means that it uses a single thread to handle all incoming requests and events. The event loop is at the core of node.js and enables asynchronous, non blocking I/O operations.

3. Event-driven Architecture:-

Node.js operates on an event-driven architecture. It allows developers to define callback functions that get executed when certain event occure, such as a when file is read, a network request completes, or a timer expire. the event driven approach is essential for non-blocking I/O operations.

4. Non-blocking I/O :-

Node.js uses non-blocking I/O operations. Which means when request is made to read or write data from a file or a network, Node.js doesn't wait for the operations to complete before moving on to the next task. instead, it continues executing other task and notified the appro-priate callback when the I/O operation is finished

5. Libuv :-

it is a multi platform C library that Node.JS use to abstract the underlying operating system's I/O compubilities and provide a consistent API for asynchronous I/O operations. it is a Responsible For managing the event loop, handling file system operations, timer, network socket and more.

6. Event queue:-

When an asynchronous is completed, its corresponding callback is placed in the event queue. The event loop continously check the event queue and execute the callback in first-in-first-out (FIFO) manner.

7. Modules:-

Node.js follow the commonJs module system, allowing developers to organize their code into reusable modules. each module encapsulates its own functionality and can be loaded into other modules using the 'require' function.

8. Concurrency model:-

Although Node.js run on single thread, it can handle high level of concurrency due to its non blocking nature. instead of creating new thread of each inloming connection, Node.js efficiently manage multiple connections using a single thread, making it highly suitable for applications that require handling a large number of concurrency request.

* Note on modules with example

=> In Node.js modules are the blocks of encapsulated code that communicate with an external application on the basis of their related functionality.

=> module can be single file or a collection of multiple files/folders.

=> The reason programmer are heavily Relavont on modules is because of their reusability as well as the ability to break down a complex piece of code into manageable chunks.

=> Three Type of modules:
1. core modules
2. Local modules
3. Third-party modules

1. Core modules:-

Node.js has many built-in modules that are part of the platform and come with Node.js installation. These module can be loaded into a program by using the required function.

- Syntax:-

```
const module = require ('module.name');
```

=> The required function will return a Javascript type depending on what a particular module return. The following ex demonstrate how to use the Node.js http module to create a web server.

- example :-

```
const http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'content-type' : 'text/html'});
    res.write('Welcome to this page');
    res.end();
}).listen(3000);
```

core modules

http
assert - set of assertion function useful for testing
fs
path
process
os
querystring
url

* local modules :-

unlike built-in and external modules, local module are created locally in your node.js application. let's create a simple calculating module that calculates various operations.

- create a calc.js file that has following code :

- calc.js

```
exports.add = function (x, y) {
        return x+y;
};
exports.sub = function (x,y) {
        return x-y;
};
exports.mult = function (x, y) {
        return x*y;
};
exports.div = function (x,y) {
        return x/y;
};
```

index.js

```
const calculator = require ('./calc');
let x=50 , y=10;
console.log (" Addition of 50 and 10 is "
                +calculator.add (x,y));
console.log (" Substraction of of 50 and 10 is "
                +calculator.sub(x,y));
console.log (" multiplication of 50 and 10 is "
                +calculator.mult (x,y));
console.log (" Division of 50 and 10 is "
                +calculator.div(x,y));
```

- Third-party modules:-

third-party modules are modules that are available online using the Node package manager (NPM). These modules can be installed in project folder or globally.

=) some of the popular third-party modules are mongoose, express, angular and React.

- example
  - npm install express
  - npm install mongoose
  - npm install -g @angular/cli

* Note on package with example

=) NPM (Node package manager)

=) it is a default package manager for Node.js and is written entirely in JavaScript.

=) Develop by Isaac Z. Schlueter in Jan 12, 2010.

=) NPM manage all the packages and modules for node.js and consist of command line client npm. It gets installed into the system with installation of Node.js.

=) Required module and package install by npm.

=) A package contain all the files needed for a module and modules are the JS libraries that can be included in Node project according to the requirement of the project.

=) NPM can install all the dependencies of a project through the package.json files. It can also update and uninstall packages.

- some facts about NPM:

=> At time of writing article, npm has 580096 registered package. the average rate of growth of this number is 291/day which outrules every other package registry.

=> NPM is open source.

=> The top npm package in the decreasing order are: lodash, async, react, request, express

- example:

npm i upper-case

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

OUTPUT

:. HELLO

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

=) Both package.json and package-lock.json are essential files used in Node.js project for managing dependencies. They are serve different purpose, and understanding their roles is crucial for proper dependency management.

1. Package.json

=) The package.json files is the heart of a Node.js project. it is a JSON file that contain metadata about the project and it's dependencies. it's typically located in root directory of the project. some of the key information stored in 'package.json' are :

- Project Information : Name, version, desc, author license, etc.
- Dependencies : A list of external package required by the project to run correctly. These dependencies are specified with their package name and version.
- DevDependencies :- similar to Dependencies, but these packag are only required during development not for the production runtime.
- script :- custome script that can be execute using npm commands.
- configuration : any specific configuration related to the project or its dependencies.

2. package-lock.json

=> The package-lock.json file is generated automatically by npm when you run the 'npm install' command.

=> It is used to lock down the version of the installed package and their dependencies. The file ensures that the exact same version of packages are installed consistently across different environment.

- The main purpose of 'package-lock.json' are:

- Dependency version locking:
  it guaranty that the specific version of package used during development are identical all team member and in production. This eliminates potential issue cause by different package versions.

- Faster and Reliable installation:-
  When 'npm install' is executed, npm reads 'package-lock.json' to determine the exact versions of package to install, which speed up the installation process and ensure reliability.

security:

The lockfile includes checksums for each installed package, which helps to verify package integrity and enhance security by mitigating the risk of unauthorized package modification.

Use and working of following Nodejs packages

URL

- Use: The url module provides utilities for parsing and formating URLs. it allow you to work with URLs, Querystring and URL components.
- important properties and methods:
  - url.parse :- parse url string and return object
  - url.format :- Formats an object into a URL string
  - url.resolve :- Resolve target URL relative to base URL.

- Parsing url and formatting URLs:

```
const url = require('url');
const urlString = 'http://www.exampl.com:8080/path?
                    query=string';
const parseUrl = url.parse(urlString);
console.log(parseUrl);
const formatUrl = url.format(parseUrl);
console.log(formatted(URL));
```

2. Process

- Use: The process module provides information and control over the current Node.js process. It allow you to interact with the process environment, argument, standard, stream & more.
- imp property and method:
  - process.argv: An array containing the command-line argument passed to the Node.js process.
  - process.env: An object containing the user environment.
  - process.exit: Terminates the Node.js process with an optional exit code.
  - process.on: Adds a listner function for a specific event.
- Program

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

3. PM2

- Use: PM2 is an external package used to manage and monitor Node.js application is production. it provides feature like process management, clustering, zero-downtime reload and log handling.
- Imp Feature:
  - process management: starting, stopping, managing Node.js processes.
  - clustering: scaling application across multiple corer for improved perfomance.
  - zero-downtime reload: allow updating application without any downtime.
  - log handling: capture, manage and route app log.

- Program:
  - npm install -g pm2
  - pm2 start app.js

4. readline
- use: The 'readline' module provides an interface for reading data from a readable stream line-by-line
- Important properties & method:
  - readline.createInterface: create a new readline interface.
  - rl.on :- used a listner function for a specific event on the readline interface.
- Program:

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

5. fs
- use: The 'fs' module provides a file system-related functionality, allowing you to read, write and manipulate files and directories.

- important properties and methods:

- fs.readFile: Asynchronously reads the contents of a file.
- fs.writeFile: Asynchronously write data to a file.
- fs.readdir: Asynchronously reads the contents of a directory.

- program

```
const fs = require('fs');
fs.readFile('file.txt', 'utf-8', (err, data) => {
    if(err) throw err;
    console.log(data);
});
fs.writeFile('newFile.txt', 'Hellow, Node.js!',
    'utf8', (err) => {
        if(err) throw err;
    console.log('File written successfully!');
});
```

6. events
- use: The 'events' module provides an event-driven architecture to handle and emit event within Node.js applications.
- imp properties & methods:

- events.EventEmitter: The class that enable event handeling and emmiting.
- emmiter.on(event): Adds a listner function for a specific event on an EventEmitter instance.
- emmiter.emit: Emits an event with optional data arguments.

• Program:

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

class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmmiter.on('customEvent', (data) => {
    console.log('Event data:' data);
});
myEmitter.emit('customevent', {message: 'custom
                event triggered!'});
```

7. Console

• Use: The console module provides methods to write to output to console or other writable streams

• imp:

- console.log:- write message
- console.error:- " error message
- console.time:- start timer
- console.~~time~~timeEnd:- stop timer

• Program

```
console.log('Hello, world!');
console.error('An error');
console.time('myTimer');
console.timeEnd('myTimer');
```

8. Buffer

• Use: it is provide a way to Handle binary data in Node.js, such as reading from stream or manipulated raw data.

• imp
  • Buffer.From : create new Buffer object From a given input data.
  • buf.toString : decodes and returns the data in the buffer object as a string
  • buf.length : return the number of bytes in Buffer object.

• Program :

```
const buf = Buffer.from('Hello', 'utf-8');
console.log(buf.toString());
console.log(buf.length);
```

9. QueryString
  • Use: it module provides utilities to work with query strings, which are used to encode URL parameters.
  • imp
    • QueryString.parse : parse a query string and return an object
    • QueryString.stringify: convert an object to a query string.
  • Program

```
const QueryString = require('QueryString');
const  ''  = 'name=John&age=30';
const parseObj = QueryString.parse(QueryString);
console.log(parseObj);

const objQueryString = {name: 'Bill', city: 'NY'};
const formattedString = QueryString.stringify(objToQuery
                                              string);
console.log(formattedString);
```

10. http

- use: it module provides functionality for creating http server and making http request.
- imp
  - http.createServer : create http server instance
  - http.Server : the class representing http server.
  - http.request : send request to a server.
- program:

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.writeHead(200, {'content-type': 'text/html'});
  res.end('Hello');
});
server.listen(8080, () => {
  console.log('Server Running on 8080');
});
```

11. V8

- use: it module exposes APIs to access the V8 JS engine internal functionality, such as memory and CPU profiling.
- imp
  - v8.getHeapStatistics() : return an object with statistics about the v8 heap memory usage.
  - v8.getHeapSpaceStatistics() : return an array of object with statistics for each space in the v8 heap.
- program:

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

12. os

•use: it is module provides information about the operating system and related functions.

•imp:

•os.cpus(): return an array of object containing information about each CPU core.

•os.totalmem(): return the total system memory in bytes.

•os.freemem(): return the free memory in bytes.

•program

```
const os = require('os');
const cpus = os.cpus();
console.log(cpus);
const totalMemory = os.totalmem();
  freeMemory = os.freemem();
console.log('Total memory : ${totalMemory} bytes');
console.log('Free memory: ${freeMemory} bytes');
```

13. zlib

•use: it provide compression and decompression functionalities for data stream using gzip, deflate and other algorithm.

•program

•zlib.creategzip: create gzip object to compress data.

•zlib.createGunzip: create a Gunzip object to decompress data.

- Program:

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

const input = fs.createReadStream('input.txt');
const output = fs.createWriteStream('input.txt.gz');

input.pipe(zlib.createGzip()).pipe(output);
```