FOREVER
Date: / / Page

Name :- Khokhariya Vishwa B.

Roll No :- 023

Subject :- 701 Date :- 19-07-23

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

* Introduction :-

- Nodejs a server-side platform.
- It uses an open source, cross-platform runtime environment for developing server-side and networking application.
- Nodejs application are written in javascript and can be run within the Nodejs runtime on OS X, microsoft windows and linux.
- Nodejs also provides a rich library of various javascript module which simplifies the development of web application using Nodejs to a great extent.

FOREVER
Page
Date: / /

Node.js = Runtime Environment
+
JS library

* Feature

- Following are some of the important Feature that make Node.js the first choice of software architects.

① Asynchronous and event Driven.

- All APIs of Nodejs library are async that is non-blocking.

- It essentially means a Nodejs 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 event of Nodejs helps the server to get a response From the previous API call.

(2) Very Fast.

- Being built on google Chrome's V8 Javascript Engine, Node js library is very fast in code execution.

(3) Single Threaded but Highly Scalable

- Node.js uses a single threaded model with event looping. Event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional server which create limited thread to handle request.

(4) No Buffering

- Node.js application never buffer any data.

- This application simply output data in chunks.

✱ Execution Architecture

- Node.js Follows an event-driven, single threaded architecture.

FOREVER
Page
Date: / /

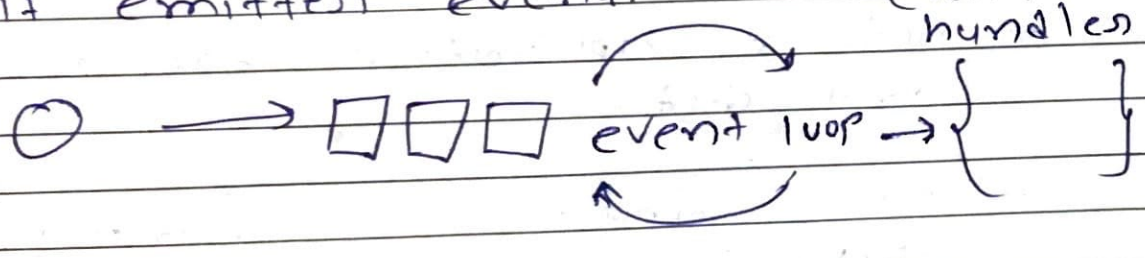
- When a Node.js application starts, it initializes an event loop that continuously listens for event and trigger appropriate callback when an event occurs.

- The event loop enables Node.js to handle multiple concurrent connection efficiently.

- It uses an event driven Proyraming model based on the Event Emitter class, where objects that emit events are known as EventEmitter.

- These events can be asynchronous op, such as File I/O, network requests, custom events triggered by the application.

- event emitter event.



- When an event occurs, Node.js executes the associated callback function, allowing the application to respond to an event

FOREVER
Page
Date: / /

- since I/O operation in Nodejs are non-blocking, the event loop can continue processing other events while waiting for I/O operations to complete.

- To handle CPU-intensive tasks without blocking the event loop. Nodejs provides a feature called child processes.

- It allows spawning additional Node.js processes to offload heavy computation and leverage multiple CPU cores.

Q-2 Note on module with example

- modules are a fundamental concept used to organize and share code.

- A module is essentially a reusable block of code that encapsulates related functionality.

- It can be a single files or a directory containing multiple files.

FOREVER
Page
Date: / /

- Node.js provides a built-in module system that allows you to create, import and use module in your application.

- This module system follows the commonjs module specification, which is the standard used in Node.js.

- example :- mathUtils.js

- const add = (a, b) => {
    return a+b;
};

- const substruct = (a, b) => {
    return a-b;
};

- module.exports = {
    add, substruct
};

- In this File we exports to module using module.exports objects which makes them accesible to other parts of our application

- int app.js File, we can import mathUtils.js

FOREVER
Date: / / Page

- const mathUtils = require('./mathUtils');
- const result1 = mathUtils.add(5,3);
console.log(result1);
- const result2 = mathUtils.substract(10,7);
console.log(result2);
- To module system allows you to organize your code into reusable units, making it easier to manage and maintain large-scale Node.js application.

Q-3 Note on package with example.

- A package refers to a collection of reusable code, usually grouped together in a directory and distributed as a compressed archive files.
- Package can contain modules, dependencies, configuration files, and other resources that make it easier to share and use code across different project.

FOREVER
Page
Date: / /

- The primary tool for managing packages in Node.js is npm, which comes bundled with node.js installation. npm provide a vast registry of packages and offers commands to install, updates and publish package.

* package.json.

- The package.json file is a vital component of any Node.js package.
- It serves as a manifest file containing metadata about the package, its dependencies, scripts and other configuration settings.
- It allows you to define project-specific details and manage dependencies efficiently.
- to install package:-

npm install <package-name>

ex npm install express.

- use of package :-

```
const express = require('express');
const app = express();
```

```
app.get('/', (req, res) => {
    res.send('Hello, world');
});

app.listen(3000, () => {
    console.log("server started on port 3000");
});
```

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

- The package.json and package-lock.json files are essential component of Node.js project.

- They serve different purpose and play crucial role in managing dependencies and ensuring consistent builds.

* package.json:-

- This files is a manifest file that contains metaduta about your Node.js project, including its name, version, description, author, scripts, dependencies, and more. It serves the following purpose.

FOREVER
Page
Date: / /

- Dependency management
  npm install
  Script Definitions.
  npm run start / test / build
  Project configuration.

* Package-lock.json
- it is automatically generated by npm when installing or ~~rewrite~~ modifying package dependencies. It serves the following process:
  - Dependency locking
  - Dependency Resolution
  - Faster installation

FOREVER
Page
Date: / /

Q-5 npm introduction and commands with use.

- npm stands for Node Package manager.
- It is a package manager for the Node JavaScript Platform.
- npm is known as the world's largest software registry.
- npm consist of three component.
- The website allows you to find third-party packages, set up profiles and manage your package.
- The command-line interface or npm CLI that runs from a terminal to allow you to interact with npm.
- large public database for JS code.
- Commands

npm init - to initialize the project directory.

- npm install <package-name>

FOREVER
Page
Date: / /

- used to install package.
- npm install <package name> -g
  - to install executable package globally.
- npm update <package name>
  - to update any package in your project
- npm uninstall <package-name>
  - to uninstall package from project.
- npm ls
  - to list out all the package installed.
- npm search <package-name>
  - to used to search the package.
- npm install <package-name> -save
  - save package

Q-6 Describe use and working of Following Nodejs Packages.

- Important properties and methods and relevant Program.

(1) url

- The URL module splits up a web address into readable parts.
- The 'url' module provides utilities for URL resolution and parsing.
- The getter and setter implement the properties of URL objects on the class prototype and the URL class is available on the global object.

* use:-

- Parse URLs: Extract different components of a URL, such as the protocol, hostname, Pathname, query parameter, etc..
- Format URLs:- convert an object with URL components into a formatted URL string.

FOREVER Page
Date: / /

- manipulate URLs :- modify or combine parts of a URL to create new URLs

* working:

- The 'url' module works with URL string and object representing URL.

- It uses the 'url.parse()' method to parse a URL string and return an object containing its components

- similarly, it uses the 'url.format()' method to format an object with URL component into a URL string.

* Important Properties and method:

- url.parse(urlString[, parseQueryString[, slashesDenoteHost]]):

  - Parse a URL and return an object containing its components.

- urlString: The URL string into parse.

- parseQueryString: (Optional) A boolean value indicating whether to parse the query string. Default is 'false'.

FOREVER
Page
Date: / /

url.format(url object): Formats an object with URL components into a URL string.

urlobject: An object with URL components like protocol, hostname, pathname, query etc..

- Relevant program:

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

const urlString = 'http://www.
    example.com/param1=value1
    &param2=value2';

- const parseURL = url.parse(urlstring);

- console.log(parseURL);
```

FOREVER Page
Date: / /

② Process:

- built-in module which provides info. and control over the current Nodejs process.
- It allows you to access environment variables, command-line arguments, standard i/o streams and communicate with the parent process

* use:-

- access command-line argument and environment variable.
- communicate with the parent process through standard i/o stream.
- Handle process-related event like process termination.

* working:-

- It automatically available in every Nodejs app.
- It represents the current Nodejs Process and provide various properties and methods to interact with it.

FOREVER
Page
Date: / /

* Important Properties and method.

- Process.argv : an array containing the command-line argument passed to the Nodejs Process.

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

* Relavant Program :-

```
const args = Process.argv.slice(2);
console.log('command-line arguments:', args);

const nodeEnv = process.env.NODE_ENV;
console.log('NODE_ENV', NodeEnv);
```

FOREVER
Page
Date: / /

3) PM2 (External Package)

- external process manager for Nodejs application.
- provides features like process clustering, automatic restarts and monitoring. making it well-suited for managing Nodejs application in production environment.

* use :-

- Run Nodejs application as background process (daemons).
- automatically restart application on crashes or failures.
- scale application across multiple CPU cores to impress performance.

* working :-

- installed globally using npm

npm install pm2 -g

pm2 start app.js

pm2 monit.

FOREVER
Page
Date: / /

(4) readline

- built-in module that provides an interface for reading input from a readable stream line by line.
- it allows you to interact with the user through prompts and get user input in a non-blocking manner.

* use:-

- create an interface for reading input from the user line.
- Prompt the user for input and receive response
- Handle user input asynchronously without blocking the event loop.

* working:

- creates an interface that reads input from the specified readable stream and triggers event when a line is received.
- It provides methods to prompt the user and get input and these operations are asynchronous allowing other tasks to proceed while waiting for user input.

FOREVER
Page
Date: / /

* Important properties and methods:

- readline.createInterface(options):
  - create readline interfaces
  options: an object containing option for the interfaces, typically specifying the input and output stream.

- interface.question(query, callback):
  - asks a question and wait for user input.

  query: The question to be displayed to the user.

  callback: A Function that will be called with the user's response as an argument.

* Relevant 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();
  });
```

FOREVER
Page
Date: / /

(5) Fs

- It provides File system relavant functionality, allowing you to read, write and manipulate files and directories.

- it enables you to interact with the file system on your machine and perform various file-relavent operation.

* use:-

- read files and directories
- write data to files.
- manipulate files and directories, such as renaming, deleting and moving.
- create and manage directories.
- work with file system | streams for handling large files efficiently.

* working:-

- Provides both synchronous and asynchronous methods for file system operation.

- synchronous method blocks the event loop until the operation is completed, while asynchronous method are non-blocking and use callbacks or Promises to handle result.

FOREVER Page
Date: / /

* Important Properties & methods.

- fs.readfile (path [, option], callback]
  path - path to the file.

- fs.writefile (file, data [, options],
  callback)

* Relevant Program :-

- const fs = require ('fs');

fs.readfile ('input.text', 'utf-8',
  (err, data) => {
  if(err) {
    console.log (err);
    return ;
  }
  console.log ('file content :', data);

fs.writefile ('output.txt', data.toUppercase(),
  'utf-8', (err) => {
  if(err) {
    console.log (err);
    return;
  }
  console.log ('Data written');
 });

});

FOREVER
Page
Date: / /

(6) events

- Provides an event-driven architecture and implements the EventEmitter class, allowing you to create and handle custom events.

* Properties & methods:-

events.EventEmitter : allows you to work with event.
emitter.on(eventName, listener): adds a listener function)
emitter.emit(eventName[---args]);
- emits specified event.

* Relevant Program:

```
- const eventEmitter = require('event');

class MyEmitter extends EventEmitter();

const myEmitter = new MyEmitter();
myEmitter.on('greet', (name) => {
        console.log('Hello, ${name}!');
});

myEmitter.emit('greet', "vishwa");
```

FOREVER
Page
Date: / /

(7) Console

* use:

- Provides methods for writing to the standard output and standard error streams, making it useful for debugging and logging.

* Properties & methods:

- console.log(message)
  :- write a message to the standard output.

- console.error(message)
  : Write an error message to standard error.

* Relevant Program:

```
console.log("Hello! log message");
console.error("Hello! error message");
```

FOREVER
Page
Date: / /

a) buffer

* use:

- provides a way to handle (; binary) binary data, allowing you to manipulate and work with raw data.

* Properties & methods

- buffer.from(data): create a new buffer from the provided data.

- buffer.tostring([encoding[, start[, end]]]);
  - return the string representation of a buffer.

* Relevant program

- const buffer = buffer.from('Hello, world', 'utf-8');

  console.log(buffer.tostring('utf8'));

FOREVER
Page
Date: / /

9) querystring :-

* use:

- Provides utilities for working with URL, querystring, parsing and formatting them.

* Properties & methods

- querystring.parse(str[, sep[, eq[, opti]]])
- querystring.stringify(obj[, sep[, eq, [,opti]]])

* Relevant program:

```
- const qs = require('querystring');
- const qs = 'param1 = value1 & param2 =
                value2';
  const parsedQ = querystring.parse(qs);
  console.log(parsedQ);
```

10) http

* use:

- Provides HTTPS server and client implementation, allowing you to build HTTP-based app.

FOREVER
Page
Date: / /

* Properties & methods

- http.createServer([requestListener]);
http.get(option[, callback]);

* Relevant Program:

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

const server = http.createServer(
    (req, res) => {
  res.writeHead(200, {'Content-Type':
                      'text/plain'});
  res.end('Hello World');
});

server.listen(3000);
```

ii) V8

- Provides access to v8 engine-specific feature and information.
- Javascript engine developed by google and used in Nodejs to execute JS code.

* use :-

- Access v8 engine stastics and information.

FOREVER
Page
Date: / /

- inspect memory usage and garbage collection metrics.

* Working

- allows you to interact with the underlying V8 engine through its APIs and retrive various performance related information and metrics

* Important Properties and methods.

V8.getHeadstatistics() : return object containing total heap size, used heap size and heap limits.

V8.getHeadspacestatistics():

* Relevant Program

```
const V8 = require('v8');
const heapstats = V8.getHeadstatistics();
console.log(heapstats);
```

FOREVER Page
Date: / /

12) OS

- Provides operating system related utilies, allowing you to access information about the underlying OS.

* use:-

- retrive info about os such as Platform, architecture and hostname.
- get info about system's CPU, network interfaces and memory.
- Handle and ~~manipulate~~ manipulate file Path.

* Important properties & methods

os.platform()
os.arch().
os.hostname()

* Relevant Program

- const os = require('os');

console.log(os.platform);

console.log("architecture", os.arch());

console.log("hostname", os.hostname());

FOREVER
Date: / / Page

14) zlib

- Provides compression and decompression functionality using the zlib library.

* use:-

- Compress data using gzip, deflate or zlib algorithms.
- Decompress data that was provided previously compressed using gzip, deflate or zib.

* Important Properties & methods:-

- zlib.createZip(options)
- zlib.createGunzip(option)
- zlib.createDeflate(option)
- zlib.createInflate(option)
- zlib.createDeflateRaw(option)
- zlib.createInflateRaw(option)
- zlib.createUnzip(option)

* Relevant Program.

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

const dataToCompress = 'This is
    some data to compress
    and decompress.';
```

```
zlib.gzip(dataToCompress, (err,
        compressedData) => {
  if(err)
{
  console.error('compression error:'
                    err);
    return;
}

console.log('compressed data: compressedData)

zlib.gunzip(compressedData, (err,
        decompressedData) => {
  if(err)
{
  console.error('Decompression
        error:', err);
    return;
}

console.log('Decompressed data:',
        decompressedData.toString());

});

});
```