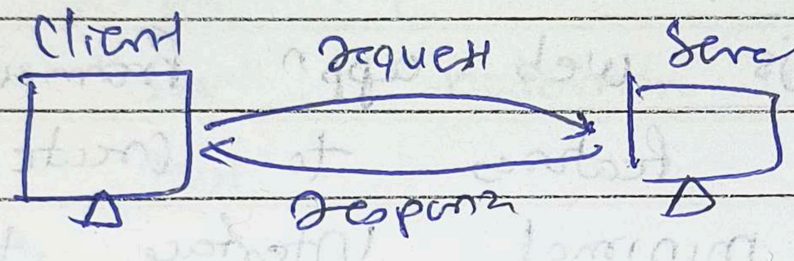
Backend Development.

Date:- / /
Page No.:-

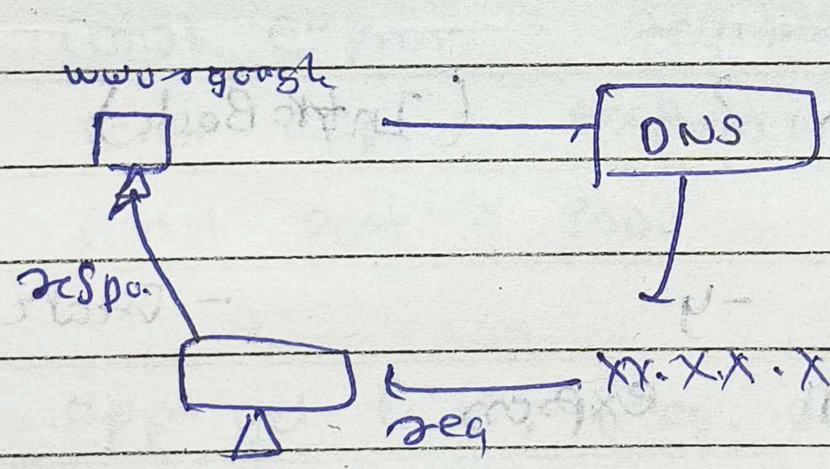
- It is process of building & maintaining the Server - side of a website or app.

- Client - Server Architecture.

- It is a Comp. n/w design where multiple clients request & receive services from centralized server



- DNS : Domain Name System.
  - Comp. that translates domain names into IP addr.



- Node.js
  - It is an open source Server environment.
  - It allows you to run JS on Server.
  - It allows developers to create web app^n & Server - side Scripts.

• Why Node.js is Fast ?

- B'coz of its asynchronous, non-blocking I/O model, which allows handle multiple concurrent requests without waiting for each one to finish.

• Express.JS.

- It is node.js web app^n framework that provides broad features to create / build web app^n
- It provides minimal interface to build our app^n.
- It is flexible as there are numerous modules available on npm, which can be directly plugged into Express.

• How to Run ? (IntroBack)

- npm init -y - Create JS file
- npm install express
- node index.js.

• framework :
- It is a set of pre-written code that provides a structure for building an application.

• Library :
- It is a collection of pre-built tools that can be used for specific task.

Harry Backend

Date:- / /
Page No.:-

Node JS. (Node.js intro)

- To run JS code in terminal

  node ./server.js → file name

- npm init
  - It is used to create a Node Project.
  - It is used in folder in which the user wants to create project.
  - The command line gives like, author, name of pr, license, scripts, description, keywords etc.

- Using this command package.json file added the starting metadata about pr, its dependency & scripts.

- npm install slugify → npm packages
  - It install node modules.

- npm init -y : It is not asked questions (nodejs-2)

- npm install --global nodemon : It monitors your project directory & automatically restarts your node app^n when detects any changes

Date:- / /
Page No.:-

- nvm : used to quickly install & use diff^n versions of node

- import & export

export const a = 1 ... file1

import {a} from "./file1";

- require

const a = require('file1')

- Working with fs & path (Nodejs - 37

const fs = require("fs")

fs.writeFile("hary.txt", "I am a boy", () => {
    fs.readFile("hary.txt", (error, data) => {
    console.log(error, data);
});
});

- Promises

import fs from "fs/promises"

let a = await fs.readFile();

Date:- / /
Page No.:-

- import path from "path";

let mypath = "./any url path";
console.log (mypath)
Console.log (path.extname (mypath))
.dirname
basename

Console.log (path.join (url)).

Expess JS.

- It is used to build single page, multi page,

1. What is NodeJS?

NodeJS is an open-source, cross-platform JavaScript runtime environment engine used for executing JavaScript code outside the browser. It is built on Google Chrome's V8 JavaScript engine. Some of the key features of NodeJS are mentioned below:

- Single-threaded
- Non-Blocking, Asynchronous I/O
- Cross-platform
- Fast Execution (V8 Engine)
- Real-Time Data Handling

2. What is NPM?

NPM stands for the Node Package Manager. It is the package manager for the NodeJS environment. It is used to install, share, and manage dependencies (libraries, tools, or packages) in JavaScript applications. Below are the following key points about the NPM:

- NPM uses a package.json file in NodeJS projects to track project dependencies, versions, scripts, and metadata like the project's name and version.
- NPM is accessed by a command-line interface (CLI). Common commands include npm install to install packages, npm update to update them, and npm uninstall to remove them.

3. Why is NodeJS single-threaded?

NodeJS is single-threaded because it's based on the asynchronous, non-blocking nature of JavaScript. This design makes it simpler to develop and maintain, and it allows NodeJS to handle many concurrent requests efficiently.

4. If NodeJS is single-threaded, then how does it handle concurrency?

NodeJS is single-threaded, but it can handle concurrency efficiently through its event-driven, non-blocking I/O model. While the event loop in NodeJS runs on a single thread, it doesn't block the execution of other tasks when waiting for I/O operations, such as file reads or database queries. Instead, NodeJS delegates these I/O tasks to the system's kernel, allowing it to continue processing other requests. Once the I/O operation is complete, the corresponding callback is added to a queue and processed by the event loop. This non-blocking approach enables NodeJS to handle multiple concurrent tasks without waiting for each one to finish sequentially.

5. Why is NodeJS preferred over other backend technologies like Java and PHP?

Here are some reasons why NodeJS is preferred:

- **Fast Performance**: NodeJS is known for its speed in handling I/O-heavy tasks.
- **NPM Ecosystem**: Node Package Manager offers over 50,000 bundles to help developers speed up development.
- **Real-Time Applications**: Perfect for data-intensive, real-time apps as it doesn't wait for APIs to return data.
- **Unified Codebase**: The Same code is used for both server and client, improving synchronization.
- **Easy for JavaScript Developers**: Since NodeJS is based on JavaScript, web developers can easily integrate it into their projects.

6. What is the difference between Synchronous and Asynchronous functions?

Here we have a difference table between Synchronous and Asynchronous functions.

Synchronous Functions	Asynchronous Functions
Blocks the execution until the task completes.	Does not block the execution; allows other tasks to proceed concurrently.
Executes tasks sequentially; each task must be completed before the next one starts.	Initiate tasks and proceed with other operations while waiting for completion.
Returns the result immediately after completion.	Typically returns a promise or callback or uses event handling to handle the result upon completion.
Errors can be easily caught with try-catch blocks.	Error handling is more complex and often involves callbacks, promises, or async/await syntax.

- **Unified Codebase**: The same code is used for both server and client, improving synchronization.
- **Easy for JavaScript Developers**: Since NodeJS is based on JavaScript, web developers can easily integrate it into their projects.

6. What is the difference between Synchronous and Asynchronous functions?

Here we have a difference table between Synchronous and Asynchronous functions.

Synchronous Functions	Asynchronous Functions
Blocks the execution until the task completes.	Does not block the execution; allows other tasks to proceed concurrently.
Executes tasks sequentially; each task must be completed before the next one starts.	Initiate tasks and proceed with other operations while waiting for completion.
Returns the result immediately after completion.	Typically returns a promise or callback or uses event handling to handle the result upon completion.
Errors can be easily caught with try-catch blocks.	Error handling is more complex and often involves callbacks, promises, or async/await syntax.
Suitable for simple, sequential tasks with predictable execution flow.	Ideal for I/O-bound operations, network requests, and tasks requiring parallel processing.

7. What are the module in NodeJS?

In a NodeJS Application, a Module can be considered as a block of code that provides 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. They are useful because of their reusability and ability to reduce the complexity of code into smaller pieces. **Examples of modules are.** http, fs, os, path, etc.

8. What is the purpose of the 'require' keyword in NodeJS?

The require keyword in NodeJS is used to include and import modules (external or built-in) into a NodeJS application.

Example

```
const http = require('http')   //imports the HTTP module to create a server.
```

9. What is middleware?

Middleware is the function that works between the request and the response cycle. Middleware gets executed after the server receives the request and before the controller sends the response.

10. How does NodeJS handle concurrency even after being single-threaded?

NodeJS handles concurrency by using asynchronous, non-blocking operations. Instead of waiting for one task to complete before starting the next, it can initiate multiple tasks and continue processing while waiting for them to finish, all within a single thread.

11. What is control flow in NodeJS?

Control flow in NodeJS refers to the sequence in which statements and functions are executed. It manages the order of execution, handling asynchronous operations, callbacks, and error handling to ensure smooth program flow.

12. What do you mean by event loop in NodeJS?

The event loop in NodeJS is a mechanism that allows it to handle multiple asynchronous tasks concurrently within a single thread. It continuously listens for events and executes associated callback functions.

13. What is the order in which control flow statements get executed?

The order in which the statements are executed is as follows:

- Execution of all inline functions

manages the order of execution, handling asynchronous operations, callbacks, and error handling to ensure smooth program flow.

12. What do you mean by event loop in NodeJS?

The event loop in NodeJS is a mechanism that allows it to handle multiple asynchronous tasks concurrently within a single thread. It continuously listens for events and executes associated callback functions.

13. What is the order in which control flow statements get executed?

The order in which the statements are executed is as follows:

- Execution and queue handling
- Collection of data and storing it
- Handling concurrency
- Executing the next lines of code

14. What are the main disadvantages of NodeJS?

Here are some main disadvantages of NodeJS listed below:

- **Single-threaded nature:** It may not fully utilize multi-core CPUs, limiting performance.
- **NoSQL preference:** Relational databases like MySQL aren't commonly used.
- **Rapid API changes:** Frequent updates can introduce instability and compatibility issues.

15. What is REPL in NodeJS?

REPL in NodeJS stands for Read, Evaluate, Print, and Loop. It is a computer environment similar to the shell which is useful for writing and debugging code as it executes the code in on go.

- **Read:** It reads the input provided by the user (JavaScript expressions or commands).
- **Eval:** It evaluates the input (executes the code).
- **Print**: It prints the result of the evaluation to the console.
- **Loop**: It loops back, allowing you to enter more code and get immediate results.

16. How to import a module in NodeJS?

We use the require module to import the External libraries in NodeJS. The result returned by require() is stored in a variable, which is used to invoke the functions using the dot notation.

17. What is the difference between NodeJS and Angular?

NodeJS	Angular
Server-side runtime environment	Front-end framework
JavaScript (or TypeScript)	TypeScript (primarily, but supports JavaScript)
Runs on the server to handle requests and responses.	Runs on the client side (browser) to build user interfaces.
Highly efficient for I/O operations due to its non-blocking nature.	Performance-focused with optimization for large-scale single-page applications.
Primarily used for server-side JavaScript execution and backend services.	Used for building interactive, dynamic, and responsive front-end applications.

18. What is package.json in NodeJS?

package.json in NodeJS is a metadata file that contains project-specific information such as dependencies, scripts, version, author details, and other configuration settings required for managing and building the project.

Example:

```
{
    "name": "app",
    "version": "1.0.0",
    "main": "index.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
```

18. What is package.json in NodeJS?

package.json in NodeJS is a metadata file that contains project-specific information such as dependencies, scripts, version, author details, and other configuration settings required for managing and building the project.

Example:

```
{
    "name": "app",
    "version": "1.0.0",
    "main": "index.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "description": "",
    "dependencies": {
        "express": "^4.21.2"
    }
}
```

19. How to create the simple HTTP server in NodeJS?

You can create a simple HTTP server in NodeJS using the built-in http module:

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

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

Output:

```
node app.js
```

Hello, World!

NodeJS

20. What are the most commonly used libraries in NodeJS?

There are the two most commonly used libraries in NodeJs:

- **ExpressJS:** ExpressJS is a minimal and flexible web application framework for building robust APIs and web apps. It simplifies routing, middleware handling, and request/response management.
- **Mongoose:** An Object Data Modeling (ODM) library for MongoDB and NodeJS, it helps in managing data relationships, schema validation, and business logic.

21. What are promises in NodeJS?

A promise is an advancement of callbacks in NodeJS. In other words, a promise is a JavaScript object that is used to handle all the asynchronous data operations. While developing an application, you may encounter that you are using a lot of nested callback functions, which causes a problem of callback hell. Promises solve this problem of callback hell.\

22. How do you install, update, and delete a dependency?

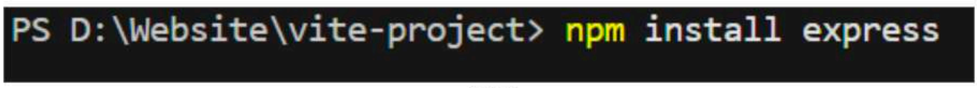
- **Install the Dependencies**

```
PS D:\Website\vite-project> npm install express
```

application, you may encounter that you are using a lot of nested callback functions, which causes a problem of callback hell. Promises solve this problem of callback hell.\

22. How do you install, update, and delete a dependency?

- **Install the Dependencies**



NodeJS

- **Update the Dependencies**

NodeJS

- **Delete the Dependencies**

NodeJS

23. Which command used to import external libraries?

In NodeJS, you can import external libraries (also known as packages) using the **`require()`** function. This allows you to include modules or packages that you have installed via **npm** (Node Package Manager).

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

NodeJS Interview Questions for Intermediate

In this set, we will be looking at intermediate Node Interview Questions for candidates with over 2 years of experience.

24. What is event-driven programming in NodeJS?

Event-driven programming is used to synchronize the occurrence of multiple events and to make the program as simple as possible. The basic components of an Event-Driven Program are:

- A callback function (called an event handler) is called when an event is triggered.
- An event loop that listens for event triggers and calls the corresponding event handler for that event.

25. What is a buffer in NodeJS?

The Buffer class in NodeJS is used to perform operations on raw binary data. Generally, Buffer refers to the particular memory location in memory. Buffer and array have some similarities, but the difference is that array can be any type, and it can be resizable. Buffers only deal with binary data, and it can not be resizable. Each integer in a buffer represents a byte. console.log() function is used to print the Buffer instance.

26. What are streams in NodeJS?

In NodeJS, streams are a powerful way to handle data in chunks rather than loading the entire data into memory. Streams allow for the efficient processing of large volumes of data, especially in situations where the data size is too large to fit into memory all at once.

There are four types of the Streams:

- **Readable Streams:** These streams allow you to read data. For example, reading data from a file or receiving HTTP request data. **Example:**

```
fs.createReadStream() or http.IncomingMessage.
```

- **Writable Streams:** These streams allow you to write data. For example, writing data to a file or sending HTTP response data. **Example:**

```
fs.createWriteStream() or http.ServerResponse.
```

- **Duplex Streams:** These are both readable and writable. You can both read and write data using the same stream. **Example**: A TCP socket.

- **Transform Streams:** These are a type of duplex stream where the data is transformed as it is read and written. **Example:** A zlib stream to compress or decompress data.

to print the Buffer instance.

26. What are streams in NodeJS?

In NodeJS, streams are a powerful way to handle data in chunks rather than loading the entire data into memory. Streams allow for the efficient processing of large volumes of data, especially in situations where the data size is too large to fit into memory all at once.

There are four types of the Streams:

- **Readable Streams:** These streams allow you to read data. For example, reading data from a file or receiving HTTP request data. **Example:**

```
fs.createReadStream() or http.IncomingMessage.
```

- **Writable Streams:** These streams allow you to write data. For example, writing data to a file or sending HTTP response data. **Example:**

```
fs.createWriteStream() or http.ServerResponse.
```

- **Duplex Streams:** These are both readable and writable. You can both read and write data using the same stream. **Example**: A TCP socket.

- **Transform Streams:** These are a type of duplex stream where the data is transformed as it is read and written. **Example:** A zlib stream to compress or decompress data.

27. Explain the crypto module in NodeJS.

The crypto module is used for encrypting, decrypting, or hashing any type of data. This encryption and decryption basically help to secure and add a layer of authentication to the data. The main use case of the crypto module is to convert the plain readable text to an encrypted format and decrypt it when required.

28. What is callback hell?

Callback hell is an issue caused by a nested callback. This causes the code to look like a pyramid and makes it unable to read To overcome this situation, we use promises.

29. Explain the use of the timers module in NodeJS.

The Timers module in NodeJS contains various functions that allow us to execute a block of code or a function after a set period. The Timers module is global, we do not need to use require() to import it.

It has the following methods:

1. setTimeout() method

The setTimeout() function is used to execute a function once after a specified delay (in milliseconds).

```
setTimeout(callback, delay, [arg1, arg2, ...]);
```

- callback: The function to be executed after the delay.
- delay: The time in milliseconds after which the function is executed.
- [arg1, arg2, ...]: Optional arguments that can be passed to the callback function.

2. setImmediate() method

The setImmediate() function is used to execute a callback function immediately after the current event loop cycle, i.e., after the I/O events in the NodeJS event loop have been processed. It is similar to setTimeout() with a delay of 0 milliseconds, but it differs in terms of when the function is executed.

```
setImmediate(callback, [arg1, arg2, ...]);
```

- callback: The function to be executed.
- [arg1, arg2, ...]: Optional arguments to pass to the callback function.

3. setInterval() method

The setInterval() function is used to execute a function repeatedly, with a fixed time delay between each call.

2. setImmediate() method

The setImmediate() function is used to execute a callback function immediately after the current event loop cycle, i.e., after the I/O events in the NodeJS event loop have been processed. It is similar to setTimeout() with a delay of 0 milliseconds, but it differs in terms of when the function is executed.

```
setImmediate(callback, [arg1, arg2, ...]);
```

- callback: The function to be executed.
- [arg1, arg2, ...]: Optional arguments to pass to the callback function.

3. setInterval() method

The setInterval() function is used to execute a function repeatedly, with a fixed time delay between each call.

```
setInterval(callback, delay, [arg1, arg2, ...]);
```

- callback: The function to be executed repeatedly.
- delay: The time in milliseconds between each execution.
- [arg1, arg2, ...]: Optional arguments that can be passed to the callback function.

30. Difference between setImmediate() and process.nextTick() methods

setImmediate()	process.nextTick()
Executes the callback after the current event loop cycl, but before the I/O tasks.	Executes the callback immediately after the current operation, before any I/O or timers.
Executes after I/O events and before timers.	Executes before any I/O events or timers.
Less likely to cause a stack overflow because it is queued after the current event loop phase.	n cause a stack overflow if used excessively because it executes before I/O or other operations, potentially blocking the event loop.
Used when you want to execute a callback after the event loop is done processing the current phase but before the next one starts.	Used to schedule a callback to be executed before any I/O events or timers in the current phase.
setImmediate(() => { console.log('Immediate'); });	process.nextTick(() => { console.log('Next Tick'); });

31. What are the different types of HTTP requests?

The different types of HTTP requests are mentioned below:

- **GET:** Retrieve data.
- **POST:** Create new resource.
- **PUT**: Update an entire resource.
- **PATCH**: Partially update a resource.
- **DELETE:** Remove a resource.

32. What is the difference between spawn() and fork() method?

spawn()	fork()
Used to launch a new process with a given command.	Used specifically for spawning new NodeJS processes.
Returns a ChildProcess object that allows interaction with the spawned process.	Returns a ChildProcess object with built-in support for IPC.
No built-in support for IPC (requires additional setup).	Built-in support for IPC, making it easier to communicate between the parent and child processes.
Running external commands or executing scripts (e.g., shell scripts, commands).	Running NodeJS scripts in child processes that can s[redacted] and receive messages from the parent process.

- **DELETE:** Remove a resource.

32. What is the difference between spawn() and fork() method?

spawn()	fork()
Used to launch a new process with a given command.	Used specifically for spawning new NodeJS processes.
Returns a ChildProcess object that allows interaction with the spawned process.	Returns a ChildProcess object with built-in support for IPC.
No built-in support for IPC (requires additional setup).	Built-in support for IPC, making it easier to communicate between the parent and child processes.
Running external commands or executing scripts (e.g., shell scripts, commands).	Running NodeJS scripts in child processes that can send and receive messages from the parent process.
spawn('ls', ['-lh', '/usr'])	fork('child.js')

33. Explain the use of the passport module in NodeJS

The passport module is used for adding authentication features to our website or web app. It implements authentication measure which helps to perform sign-in operations.

34. What is a, fork in NodeJS?

Fork is a method in NodeJS that is used to create child processes. It helps to handle the increasing workload. It creates a new instance of the engine which enables multiple processes to run the code.

35. What are the three methods to avoid callback hell?

The three methods to avoid callback hell are:

- Using async/await()
- Using promises
- Using generators

36. What is a .body-parser in NodeJS?

Body-parser is the NodeJS body-parsing middleware. It is responsible for parsing the incoming request bodies in a middleware before you handle it. It is an NPM module that processes data sent in HTTP requests.

37. What is CORS in NodeJS?

The word CORS stands for “Cross-Origin Resource Sharing”. Cross-Origin Resource Sharing is an HTTP-header based mechanism implemented by the browser which allows a server or an API to indicate any origins (different in terms of protocol, hostname, or port) other than its origin from which the unknown origin gets permission to access and load resources. The cors package available in the npm registry is used to tackle CORS errors in a NodeJS application.

38. Explain the tls module in NodeJS..

The tls module provides an implementation of the Transport Layer Security (TLS) and Secure Socket Layer (SSL) protocols that are built on top of OpenSSL. It helps to establish a secure connection on the network.

39. Can you access DOM in Node?

No, you cannot access the DOM in NodeJS because NodeJS is a **server-side** environment, while the **DOM (Document Object Model)** is a **client-side** concept used in browsers to interact with HTML and XML documents.

NodeJS runs on the server and does not have access to a browser's DOM, which is part of the browser's environment. The DOM allows you to manipulate the content and structure of web pages, but it is not available in NodeJS, as it operates on the backend, outside the context of a web page or browser.

40. How do you manage packages in your NodeJS project?

In a NodeJS project, package management is typically done through npm (Node Package Manager).

connection on the network.

39. Can you access DOM in Node?

No, you cannot access the DOM in NodeJS because NodeJS is a **server-side** environment, while the **DOM (Document Object Model)** is a **client-side** concept used in browsers to interact with HTML and XML documents.

NodeJS runs on the server and does not have access to a browser's DOM, which is part of the browser's environment. The DOM allows you to manipulate the content and structure of web pages, but it is not available in NodeJS, as it operates on the backend, outside the context of a web page or browser.

40. How do you manage packages in your NodeJS project?

In a NodeJS project, package management is handled through **npm (Node Package Manager)**, which is the default package manager for NodeJS, which allows you to install and manage third-party packages and create and publish your packages.

41. What is the purpose of NODE_ENV?

The `NODE_ENV` environment variable in NodeJS is used to specify the environment in which the NodeJS application is running. It helps in distinguishing between different stages of the application's lifecycle, such as development, testing, or production, and allows you to customize the behavior of the application based on that environment.

42. What is a test pyramid in NodeJS?

The **Test Pyramid** is a strategy for structuring tests in a software project to ensure efficiency, maintainability, and good coverage. It consists of three levels:

- **Unit Tests (Base)**: Test individual components or functions in isolation. These tests are fast and numerous. **Example**: Testing a single function like `add(1, 2)`.
- **Integration Tests (Middle)**: Test interactions between components to ensure they work together. These are slower than unit tests but cover more functionality. **Example**: Testing API routes to ensure they connect properly with the database.
- **End-to-End Tests (Top)**: Test the entire application flow from the user interface to the backend. These are slow and fewer in number. **Example**: Simulating user login and navigating the application.

NodeJS Interview Questions for Experienced

In this set, we will be covering Node interview question for experienced developers with over 5 years of experience.

43. What is piping in NodeJS?

In NodeJS, **piping** refers to the process of passing the output of one stream directly into another stream. It allows data to flow through multiple streams without needing to store it in memory or temporarily write it to disk. This is a common pattern used in file handling, HTTP requests, and other I/O operations in NodeJS.

44. What is a cluster in NodeJS?

Due to a single thread in NodeJS, it handles memory more efficiently because there are no multiple threads due to which no thread management is needed. Now, to handle workload efficiently and to take advantage of computer multi-core systems, cluster modules are created that provide us the way to make child processes that run simultaneously with a single parent process.

45. Explain some of the cluster methods in NodeJS

- **Fork():** It creates a new child process from the master. The isMaster returns true if the current process is master or else false.
- **isWorker:** It returns true if the current process is a worker or else false.
- **process:** It returns the child process which is global.
- **send():** It sends a message from worker to master or vice versa.
- **kill():** It is used to kill the current worker.

46. How to manage sessions in NodeJS?

Session management can be done in NodeJS by using the express-session module. It helps in saving the data in the key-value form. In this module, the session data is not saved in the cookie itself, just the session ID

- **kill():** It is used to kill the current worker.

46. How to manage sessions in NodeJS?

Session management can be done in NodeJS by using the express-session module. It helps in saving the data in the key-value form. In this module, the session data is not saved in the cookie itself, just the session ID.

47. How many types of API functions are there in Node.js?

There are two types of API functions:

- **Asynchronous, non-blocking functions** - mostly I/O operations which can be fork out of the main loop.
- **Synchronous, blocking functions** - mostly operations that influence the process running in the main loop.

48. How can we implement authentication and authorization in NodeJS?

Authentication is the process of verifying a user's identity while authorization is determining what actions can be performed. We use packages like Passport and JWT to implement authentication and authorization.

49. Explain the packages used for file uploading in NodeJS.the

The package used for file uploading in NodeJS is Multer. The file can be uploaded to the server using this module. There are other modules in the market but Multer is very popular when it comes to file uploading. Multer is a NodeJS middleware that is used for handling multipart/form-data, which is a mostly used library for uploading files.

50. Explain the difference between NodeJS and server-side scripting languages like Python

NodeJS	Server-Side Scripting Languages (e.g., Python)
A runtime environment for executing JavaScript outside the browser.	A programming language that can be used for server-side scripting (e.g., Python, PHP, Ruby).
Built on JavaScript, primarily used for asynchronous programming.	Built on Python, which is synchronous by default, but can also be used for asynchronous programming.
Non-blocking, event-driven I/O model using the Event Loop.	Thread-based concurrency (multi-threading) or asynchronous programming with frameworks like asyncio.
Highly performant for I/O-heavy tasks but less efficient for CPU-heavy operations due to single-threaded nature.	More suitable for CPU-heavy operations but can be less performant in handling high concurrency compared to NodeJS.
Used for building fast, scalable, I/O-bound applications (e.g., APIs, real-time apps).	Commonly used for general-purpose programming, web development, and CPU-bound tasks (e.g., Django for web, machine learning with libraries like TensorFlow).

51. How to Connect NodeJS to a MongoDB Database?

To connect to the MongoDB database write the following code after installing the Mongoose package:

```
const mongoose = require("mongoose");

mongoose.connect("DATABASE_URL_HERE", {
    useNewUrlParser: true,
    useUnifiedTopology: true
});
```

52. How to read command line arguments in NodeJS?

Command-line arguments (CLI) are strings of text used to pass additional information to a program when an application is running through the command line interface of an operating system. We can easily read these arguments by the global object in node i.e. process object. Below is the approach:

Step 1: Save a file as index.js and paste the below code inside the file.

```
let arguments = process.argv;
```

```
mongoose.connect("DATABASE_URL_HERE", {
   useNewUrlParser: true,
   useUnifiedTopology: true
});
```

52. How to read command line arguments in NodeJS?

Command-line arguments (CLI) are strings of text used to pass additional information to a program when an application is running through the command line interface of an operating system. We can easily read these arguments by the global object in node i.e. process object. Below is the approach:

Step 1: Save a file as index.js and paste the below code inside the file.

```
let arguments = process.argv ;

console.log(arguments) ;
```

Step 2: Run the index.js file using the below command:

```
node index.js
```

53. Explain the NodeJS Redis module.

Redis is an Open Source store for storing data structures. It is used in multiple ways. It is used as a database, cache, and message broker. It can store data structures such as strings, hashes, sets, sorted sets, bitmaps, indexes, and streams. Redis is very useful for NodeJS developers as it reduces the cache size which makes the application more efficient. However, it is very easy to integrate Redis with NodeJS applications.

54. What is web socket?

Web Socket is a protocol that provides full-duplex (multiway) communication i.e. allows communication in both directions simultaneously. Web Socket is a modern web technology in which there is a continuous connection between the user's browser (client) and the server. In this type of communication, between the web server and the web browser, both of them can send messages to each other at any point in time.

55. Explain the util module in NodeJS

The Util module in NodeJS provides access to various utility functions. There are various utility modules available in the NodeJS module library.

- **OS Module:** Operating System-based utility modules for NodeJS are provided by the OS module.
- **Path Module:** The path module in NodeJS is used for transforming and handling various file paths.
- **DNS Module:** DNS Module enables us to use the underlying Operating System name resolution functionalities. The actual DNS lookup is also performed by the DNS Module.
- **Net Module:** Net Module in NodeJS is used for the creation of both client and server. Similar to DNS Module this module also provides an asynchronous network wrapper.

56. How to handle environment variables in NodeJS?

We use process.env to handle environment variables in NodeJS. We can specify environment configurations as well as keys in the .env file. To access the variable in the application, we use the "process.env.VARIABLE_NAME" syntax. To use it we have to install the dotenv package using the below command:

```
npm install dotenv
```

57. Explain DNS module in NodeJS

DNS is a node module used to do name resolution facility which is provided by the operating system as well as used to do an actual DNS lookup. Its main advantage is that there is no need for memorizing IP addresses – DNS servers provide a nifty solution for converting domain or subdomain names to IP addresses.

58. What is the difference between setImmediate() and setTimeout()?

setImmediate()	setTimeout()
Executes the callback immediately after the current event loop phase.	Executes the callback after the specified delay (in ms).

memorizing IP addresses – DNS servers provide a nifty solution for converting domain or subdomain names to IP addresses.

58. What is the difference between setImmediate() and setTimeout()?

stelmmediate()	setTimeout()
Executes the callback immediately after the current event loop phase.	Executes the callback after the specified delay (in ms).
Adds the callback to the next iteration of the event loop, in the check phase.	Adds the callback to the timer queue, to be executed after the specified delay.
Has no timer; it is designed for immediate execution.	Executes only after the specified delay, which may vary slightly based on system timing resolution.
No delay, always executes after the current phase finishes.	Executes after the specified delay (minimum of 1 ms).
Used for callbacks that need to run immediately after I/O events or timers.	Used for delaying the execution of a callback by a specific time.

59. For NodeJS, why does Google use the V8 engine?

Google **for** the V8 engine for NodeJS of the following reasons mentioned below:

- **High Performance:** V8 is a highly optimized JavaScript engine designed for speed. It compiles JavaScript directly into machine code, which makes it much faster than interpreted JavaScript.
- **Just-In-Time (JIT) Compilation:** V8 uses JIT compilation, which translates JavaScript code into machine code during execution, enabling faster execution compared to traditional interpretation.
- **Cross-platform Compatibility:** V8 is cross-platform due to which the NodeJS application can run on **that** platforms.
- **Integration with Google Chrome:** The Google Chrome by using the V8 engine in the NodeJS ensures consistency in performance and features.
- **Asynchronous I/O Efficiency: The** V8 engine can handle the non-blocking, asynchronous I/O operations which is important for handling the multiple tasks in NodeJS.

60. What is an Event Emitter in Node.js?

In Node.js, an **Event Emitter**is a class that allows objects to emit events and register listeners (callbacks) to handle those events. It is part of the **events module** and is commonly used to handle asynchronous events and to implement an observer pattern, where an object (the emitter) triggers events, and other objects (listeners) respond to those events.

Key Concepts of EventEmitter:

- **Emitting Events**: Objects can emit custom events by calling the `.emit()` method. This triggers all listeners that are registered for that event.
- **Listening to Events**: You can listen for specific events using the `.on()` or `.once()` methods. `.on()` will listen for the event indefinitely, while `.once()` will listen for the event only once.
- **Event Handling**: Event listeners (callbacks) are executed when the corresponding event is emitted. These listeners are often used to handle asynchronous operations like HTTP requests or file I/O.

Beginner Node.js Interview Questions

1. What is Node.js and how it works?

Node.js is a virtual machine that uses JavaScript as its scripting language and runs Chrome's V8 JavaScript engine. Basically, Node.js is based on an event-driven architecture where I/O runs asynchronously making it lightweight and efficient. It is being used in developing desktop applications as well with a popular framework called electron as it provides API to access OS-level features such as file system, network, etc.

Here is a Free course on Node.js for beginners to master the fundamentals of Node.js.

2. What tools can be used to assure consistent code style?

ESLint can be used with any IDE to ensure a consistent coding style which further helps in maintaining the codebase.

3. What is a first class function in Javascript?

When functions can be treated like any other variable then those functions are first-class functions. There are many other programming languages, for example, scala, Haskell, etc which follow this including JS. Now because of this function can be passed as a param to another function(callback) or a function can return another function(higher-order function). map() and filter() are higher-order functions that are popularly used.

4. How do you manage packages in your node.js project?

It can be managed by a number of package installers and their configuration file accordingly. Out of them mostly use npm or yarn. Both provide almost all libraries of javascript with extended features of controlling environment-specific configurations. To maintain versions of libs being installed in a project we use package.json and package-lock.json so that there is no issue in porting that app to a different environment.

5. How is Node.js better than other frameworks most popularly used?

- Node.js provides simplicity in development because of its non-blocking I/O and event-based model results in short response time and concurrent processing, unlike other frameworks where developers have to use thread management.
- It runs on a chrome v8 engine which is written in c++ and is highly performant with constant improvement.
- Also since we will use Javascript in both the frontend and backend the development will be much faster.
- And at last, there are sample libraries so that we don't need to reinvent the wheel.

6. Explain the steps how "Control Flow" controls the functions calls?

- Control the order of execution
- Collect data
- Limit concurrency
- Call the following step in the program.

7. What are some commonly used timing features of Node.js?

- **setTimeout/clearTimeout** – This is used to implement delays in code execution.
- **setInterval/clearInterval** – This is used to run a code block multiple times.
- **setImmediate/clearImmediate** – Any function passed as the setImmediate() argument is a callback that's executed in the next iteration of the event loop.
- **process.nextTick** – Both setImmediate and process.nextTick appear to be doing the same thing; however, you may prefer one over the other depending on your callback's urgency.

8. What are the advantages of using promises instead of callbacks?

The main advantage of using promise is you get an object to decide the action that needs to be taken after the async task completes. This gives more manageable code and avoids callback hell.

9. What is fork in node JS?

A fork in general is used to spawn child processes. In node it is used to create a new instance of v8 engine to run multiple workers to execute the code.

10. Why is Node.js single-threaded?

Node.js was created explicitly as an experiment in async processing. This was to try a new theory of doing async processing on a single thread over the existing thread-based implementation of scaling via different frameworks.

11. How do you create a simple server in Node.js that returns Hello World?

```
var http = require("http");
http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
```

single thread over the existing thread-based implementation of scaling via different frameworks.

11. How do you create a simple server in Node.js that returns Hello World?

```
var http = require("http");
http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(3000);
```

12. How many types of API functions are there in Node.js?

There are two types of API functions:

- **Asynchronous, non-blocking functions** - mostly I/O operations which can be fork out of the main loop.
- **Synchronous, blocking functions** - mostly operations that influence the process running in the main loop.

13. What is REPL?

PL in Node.js stands for **R**ead, **E**val, **P**rint, and **L**oop, which further means evaluating code on the go.

14. List down the two arguments that async.queue takes as input?

- Task Function
- Concurrency Value

15. What is the purpose of module.exports?

This is used to expose functions of a particular module or file to be used elsewhere in the project. This can be used to encapsulate all similar functions in a file which further improves the project structure.

For example, you have a file for all utils functions with util to get solutions in a different programming language of a problem statement.

```
const getSolutionInJavaScript = async ({
 problem_id
}) => {
...
};
const getSolutionInPython = async ({
 problem_id
}) => {
...
};
module.exports = { getSolutionInJavaScript, getSolutionInPython }
```

Thus using module.exports we can use these functions in some other file:

```
const { getSolutionInJavaScript, getSolutionInPython} = require("./utils")
```

Intermediate Node.js Interview Questions

16. Explain the concept of stub in Node.js?

Stubs are used in writing tests which are an important part of development. It replaces the whole function which is getting tested.

This helps in scenarios where we need to test:

- External calls which make tests slow and difficult to write (e.g HTTP calls/ DB calls)
- Triggering different outcomes for a piece of code (e.g. what happens if an error is thrown/ if it passes)

For example, this is the function:

```
const request = require('request');
const getPhotosByAlbumId = (id) => {
const requestUrl = `https://jsonplaceholder.typicode.com/albums/${id}/photos?_limit=3`;
return new Promise((resolve, reject) => {
    request.get(requestUrl, (err, res, body) => {
        if (err) {
            return reject(err);
        }
        resolve(JSON.parse(body));
    });
});
};
module.exports = getPhotosByAlbumId;
To test this function this is the stub
const expect = require('chai').expect;
const request = require('request');
```

```
const { getSolutionInJavaScript, getSolutionInPython} = require("./utils")
```

Intermediate Node.js Interview Questions

16. Explain the concept of stub in Node.js?

Stubs are used in writing tests which are an important part of development. It replaces the whole function which is getting tested.

This helps in scenarios where we need to test:

- External calls which make tests slow and difficult to write (e.g HTTP calls/ DB calls)
- Triggering different outcomes for a piece of code (e.g. what happens if an error is thrown/ if it passes)

For example, this is the function:

```
const request = require('request');
const getPhotosByAlbumId = (id) => {
const requestUrl = `https://jsonplaceholder.typicode.com/albums/${id}/photos?_limit=3`;
return new Promise((resolve, reject) => {
    request.get(requestUrl, (err, res, body) => {
        if (err) {
            return reject(err);
        }
        resolve(JSON.parse(body));
    });
});
};
module.exports = getPhotosByAlbumId;
To test this function this is the stub
const expect = require('chai').expect;
const request = require('request');
const sinon = require('sinon');
const getPhotosByAlbumId = require('./index');
describe('with Stub: getPhotosByAlbumId', () => {
before(() => {
    sinon.stub(request, 'get')
        .yields(null, null, JSON.stringify([
            {
                "albumId": 1,
                "id": 1,
                "title": "A real photo 1",
                "url": "https://via.placeholder.com/600/92c952",
                "thumbnailUrl": "https://via.placeholder.com/150/92c952"
            },
            {
                "albumId": 1,
                "id": 2,
                "title": "A real photo 2",
                "url": "https://via.placeholder.com/600/771796",
                "thumbnailUrl": "https://via.placeholder.com/150/771796"
            },
            {
                "albumId": 1,
                "id": 3,
                "title": "A real photo 3",
                "url": "https://via.placeholder.com/600/24f355",
                "thumbnailUrl": "https://via.placeholder.com/150/24f355"
            }
        ]));
});
after(() => {
    request.get.restore();
});
it('should getPhotosByAlbumId', (done) => {
    getPhotosByAlbumId(1).then((photos) => {
        expect(photos.length).to.equal(3);
        photos.forEach(photo => {
            expect(photo).to.have.property('id');
            expect(photo).to.have.property('title');
            expect(photo).to.have.property('url');
        });
        done();
    });
});
});
```

17. Describe the exit codes of Node.js?

Exit codes give us an idea of how a process got terminated/the reason behind termination.

A few of them are:

- Uncaught fatal exception - (code - 1) - There has been an exception that is nc
- Unused - (code - 2) - This is reserved by bash

```
});
```

17. Describe the exit codes of Node.js?

Exit codes give us an idea of how a process got terminated/the reason behind termination.

A few of them are:

- Uncaught fatal exception - (code - 1) - There has been an exception that is not handled
- Unused - (code - 2) - This is reserved by bash
- Fatal Error - (code - 5) - There has been an error in V8 with stderr output of the description
- Internal Exception handler Run-time failure - (code - 7) - There has been an exception when bootstrapping function was called
- Internal JavaScript Evaluation Failure - (code - 4) - There has been an exception when the bootstrapping process failed to return function value when evaluated.

18. For Node.js, why Google uses V8 engine?

Well, are there any other options available? Yes, of course, we have Spidermonkey from Firefox, Chakra from Edge but Google's v8 is the most evolved(since it's open-source so there's a huge community helping in developing features and fixing bugs) and fastest(since it's written in c++) we got till now as a JavaScript and WebAssembly engine. And it is portable to almost every machine known.

19. Why should you separate Express app and server?

The server is responsible for initializing the routes, middleware, and other application logic whereas the app has all the business logic which will be served by the routes initiated by the server. This ensures that the business logic is encapsulated and decoupled from the application logic which makes the project more readable and maintainable.

20. Explain what a Reactor Pattern in Node.js?

Reactor pattern again a pattern for nonblocking I/O operations. But in general, this is used in any event-driven architecture.

There are two components in this: 1. Reactor 2. Handler.

Reactor: Its job is to dispatch the I/O event to appropriate handlers
Handler: Its job is to actually work on those events

21. What is middleware?

Middleware comes in between your request and business logic. It is mainly used to capture logs and enable rate limit, routing, authentication, basically whatever that is not a part of business logic. There are third-party middleware also such as body-parser and you can write your own middleware for a specific use case.

22. What are node.js buffers?

In general, buffers is a temporary memory that is mainly used by stream to hold on to some data until consumed. Buffers are introduced with additional use cases than JavaScript's Unit8Array and are mainly used to represent a fixed-length sequence of bytes. This also supports legacy encodings like ASCII, utf-8, etc. It is a fixed(non-resizable) allocated memory outside the v8.

23. What is node.js streams?

Streams are instances of EventEmitter which can be used to work with streaming data in Node.js. They can be used for handling and manipulating streaming large files(videos, mp3, etc) over the network. They use buffers as their temporary storage.

There are mainly four types of the stream:

- **Writable:** streams to which data can be written (for example, fs.createWriteStream()).
- **Readable:** streams from which data can be read (for example, fs.createReadStream()).
- **Duplex:** streams that are both Readable and Writable (for example, net.Socket).
- **Transform:** Duplex streams that can modify or transform the data as it is written and read (for example, zlib.createDeflate()).

24. How can we use async await in node.js?

Here is an example of using async-await pattern:

```
// this code is to retry with exponential backoff
function wait (timeout) {
 return new Promise((resolve) => {
setTimeout(() => {
  resolve()
}, timeout);
 });
}
async function requestWithRetry (url) {
 const MAX_RETRIES = 10;
 for (let i = 0; i <= MAX_RETRIES; i++) {
try {
  return await request(url);
} catch (err) {
  const timeout = Math.pow(2, i);
  console.log('Waiting', timeout, 'ms');
  await wait(timeout);
  console.log('Retrying', err.message, i);
}
 }
```

- **Duplex:** streams that are both Readable and Writable (for example, net.Socket).
- **Transform:** Duplex streams that can modify or transform the data as it is written and read (for example, zlib.createDeflate()).

24. How can we use async await in node.js?

Here is an example of using async-await pattern:

```
// this code is to retry with exponential backoff
function wait (timeout) {
 return new Promise((resolve) => {
setTimeout(() => {
  resolve()
}, timeout);
 });
}
async function requestWithRetry (url) {
 const MAX_RETRIES = 10;
 for (let i = 0; i <= MAX_RETRIES; i++) {
try {
  return await request(url);
} catch (err) {
  const timeout = Math.pow(2, i);
  console.log('Waiting', timeout, 'ms');
  await wait(timeout);
  console.log('Retrying', err.message, i);
}
 }
}
```

25. How does Node.js overcome the problem of blocking of I/O operations?

Since the node has an event loop that can be used to handle all the I/O operations in an asynchronous manner without blocking the main function.

So for example, if some network call needs to happen it will be scheduled in the event loop instead of the main thread(single thread). And if there are multiple such I/O calls each one will be queued accordingly to be executed separately(other than the main thread).

Thus even though we have single-threaded JS, I/O ops are handled in a nonblocking way.

26. Differentiate between process.nextTick() and setImmediate()?

Both can be used to switch to an asynchronous mode of operation by listener functions.

process.nextTick() sets the callback to execute but setImmediate pushes the callback in the queue to be executed. So the event loop runs in the following manner

timers–>pending callbacks–>idle,prepare–>connections(poll,data,etc)–>check–>close callbacks

In this process.nextTick() method adds the callback function to the start of the next event queue and setImmediate() method to place the function in the check phase of the next event queue.

27. If Node.js is single threaded then how does it handle concurrency?

The main loop is single-threaded and all async calls are managed by libuv library.

For example:

```
const crypto = require("crypto");
const start = Date.now();
function logHashTime() {
 crypto.pbkdf2("a", "b", 100000, 512, "sha512", () => {
console.log("Hash:", Date.now() - start);
 });
}
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

This gives the output:

```
Hash: 1213
Hash: 1225
Hash: 1212
Hash: 1222
```

This is because libuv sets up a thread pool to handle such concurrency. How many threads will be there in the thread pool depends upon the number of cores but you can override this.

28. What is an event-loop in Node JS?

Whatever that is async is managed by event-loop using a queue and listener. We can get the idea using the following diagram:

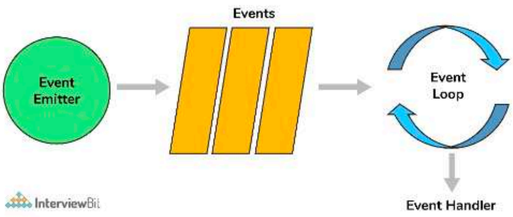
Node.js Event Loop

This is because libuv sets up a thread pool to handle such concurrency. How many threads will be there in the thread pool depends upon the number of cores but you can override this.

28. What is an event-loop in Node JS?

Whatever that is async is managed by event-loop using a queue and listener. We can get the idea using the following diagram:

Node.js Event Loop



So when an async function needs to be executed(or I/O) the main thread sends it to a different thread allowing v8 to keep executing the main code. Event loop involves different phases with specific tasks such as timers, pending callbacks, idle or prepare, poll, check, close callbacks with different FIFO queues. Also in between iterations it checks for async I/O or timers and shuts down cleanly if there aren't any.

29. What do you understand by callback hell?

```
async_A(function(){
   async_B(function(){
       async_C(function(){
           async_D(function(){
           ....
           });
       });
   });
});
```

For the above example, we are passing callback functions and it makes the code unreadable and not maintainable, thus we should change the async logic to avoid this.

Advanced Node.js Interview Questions

30. What is an Event Emitter in Node.js?

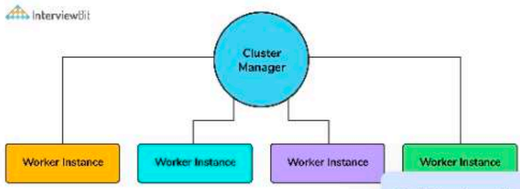
EventEmitter is a Node.js class that includes all the objects that are basically capable of emitting events. This can be done by attaching named events that are emitted by the object using an eventEmitter.on() function. Thus whenever this object throws an even the attached functions are invoked synchronously.

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
 console.log('an event occurred!');
});
myEmitter.emit('event');
```

31. Enhancing Node.js performance through clustering.

Node.js applications run on a single processor, which means that by default they don't take advantage of a multiple-core system. Cluster mode is used to start up multiple node.js processes thereby having multiple instances of the event loop. When we start using cluster in a nodejs app behind the scene multiple node.js processes are created but there is also a parent process called the **cluster manager** which is responsible for monitoring the health of the individual instances of our application.

Clustering in Node.js

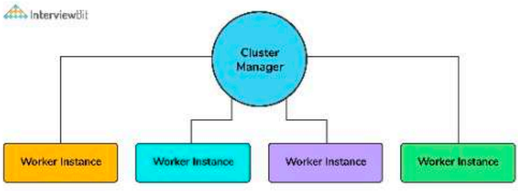


```
myEmitter.on('event', () => {
 console.log('an event occurred!');
});
myEmitter.emit('event');
```

31. Enhancing Node.js performance through clustering.

Node.js applications run on a single processor, which means that by default they don't take advantage of a multiple-core system. Cluster mode is used to start up multiple node.js processes thereby having multiple instances of the event loop. When we start using cluster in a nodejs app behind the scene multiple node.js processes are created but there is also a parent process called the **cluster manager** which is responsible for monitoring the health of the individual instances of our application.

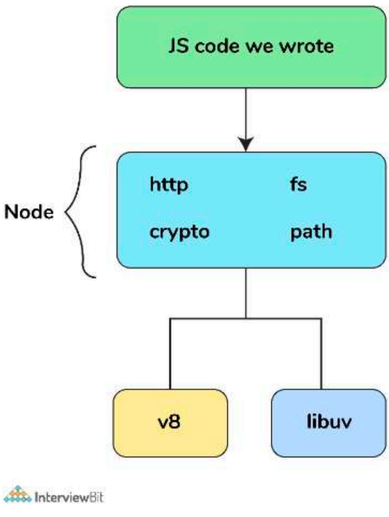
Clustering in Node.js



32. What is a thread pool and which library handles it in Node.js

The Thread pool is handled by the libuv library. libuv is a multi-platform C library that provides support for asynchronous I/O-based operations such as file systems, networking, and concurrency.

Thread Pool



33. What is WASI and why is it being introduced?

Web assembly provides an implementation of WebAssembly System Interface specification through WASI API in node.js implemented using WASI class. The introduction of WASI was done by keeping in mind its possible to use the underlying operating system via a collection of POSIX-like functions thus further enabling the application to use resources more efficiently and features that require system-level access.

34. How are worker threads different from clusters?

Cluster:

- There is one process on each CPU with an IPC to communicate.
- In case we want to have multiple servers accepting HTTP requests via a single port, clusters can be helpful.
- The processes are spawned in each CPU thus will have separate memory and node instance which further will lead to memory issues.

Worker threads:

- There is only one process in total with multiple threads.
- Each thread has one Node instance (one event loop, one JS engine) with most of the APIs accessible.
- Shares memory with other threads (e.g. SharedArrayBuffer)
- This can be used for CPU-intensive tasks like processing data or accessing the file system since NodeJS is single-threaded, synchronous tasks can be made more efficient leveraging the worker's threads.

35. How to measure the duration of async operations?

Performance API provides us with tools to figure out the necessary performance

- This can be used for CPU-intensive tasks like processing data or accessing the file system since NodeJS is single-threaded, synchronous tasks can be made more efficient leveraging the worker's threads.

35. How to measure the duration of async operations?

Performance API provides us with tools to figure out the necessary performance metrics. A simple example would be using async_hooks and perf_hooks

```
'use strict';
const async_hooks = require('async_hooks');
const {
 performance,
 PerformanceObserver
} = require('perf_hooks');
const set = new Set();
const hook = async_hooks.createHook({
 init(id, type) {
if (type === 'Timeout') {
  performance.mark(`Timeout-${id}-Init`);
  set.add(id);
}
 },
 destroy(id) {
if (set.has(id)) {
  set.delete(id);
  performance.mark(`Timeout-${id}-Destroy`);
  performance.measure(`Timeout-${id}`,
                      `Timeout-${id}-Init`,
                      `Timeout-${id}-Destroy`);
}
 }
});
hook.enable();
const obs = new PerformanceObserver((list, observer) => {
 console.log(list.getEntries()[0]);
 performance.clearMarks();
 observer.disconnect();
});
obs.observe({ entryTypes: ['measure'], buffered: true });
setTimeout(() => {}, 1000);
```

This would give us the exact time it took to execute the callback.

36. How to measure the performance of async operations?

Performance API provides us with tools to figure out the necessary performance metrics.

A simple example would be:

```
const { PerformanceObserver, performance } = require('perf_hooks');
const obs = new PerformanceObserver((items) => {
 console.log(items.getEntries()[0].duration);
 performance.clearMarks();
});
obs.observe({ entryTypes: ['measure'] });
performance.measure('Start to Now');
performance.mark('A');
doSomeLongRunningProcess(() => {
 performance.measure('A to Now', 'A');
 performance.mark('B');
 performance.measure('A to B', 'A', 'B');
});
```

Additional Useful Resources

1. NodeJS MCQ
2. Top 10 Node JS Projects Ideas
3. Node.js Vs React.js: What's The Difference?
4. Node.js Vs Django: Which One is Better For Web Development?

Node js MCQ

1.Which of the following are Node.js stream types?

- ○ Duplex
- ○ Readable
- ○ Writable
- ○ All of the above

2. Which module is used to serve static files in Node.js?