1. Explain in brief what is node js?

A. Node.js was developed by Ryan Dahl in 2009. Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications.Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.  
Node.js = Runtime Environment + JavaScript Library

## Features of Node.js

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

* Asynchronous and Event Driven − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* Very Fast − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* Single Threaded but Highly Scalable − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers 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 requests than traditional servers like Apache HTTP Server.
* No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.

2.How is node js non-blocking?

A.Node.js maintains a limited thread pool to serve requests.

* Whenever a request comes, Node.js places it into a queue.
* Now, the single-threaded “Event loop”—the core component—comes into the picture. This event loop waits for requests indefinitely.
* When a request comes in, the loop picks it up from the queue and checks whether it requires a blocking input/output (I/O) operation. If not, it processes the request and sends a response.
* If the request has a blocking operation to perform, the event loop assigns a thread from the internal thread pool to process the request. There are limited internal threads available. This group of auxiliary threads is called the worker group.
* The event loop tracks blocking requests and places them in the queue once the blocking task is processed. This is how it maintains its non-blocking nature.

Since Node.js uses fewer threads, it utilizes fewer resources/memory, resulting in faster task execution. So for our purposes, this single-threaded architecture is equivalent to multi-threaded architecture.

3.What is throughput?  
A. Throughput is a way to find the efficiency of a CPU. It can be defined as the number of processes executed by the CPU in a given amount of time. For example, let's say, the process P1 takes 3 seconds for execution, P2 takes 5 seconds, and P3 takes 10 seconds. So, throughput, in this case, the throughput will be (3+5+10)/3 = 18/3 = 6 seconds

4.How is Node js having high IO throughput?  
A.Node.js uses something known as an asynchronous I/O. When you make a request and your request wants to access some database or read from a file (which is what most web requests do), your request is put on a event loop and a callback is attached (token in our example). And then the main event thread is free to do anything, even service more requests. When I/O is complete, your callback is serviced and your response is sent.

Thus node.js inherently provides more throughput as the main thread handles every wait operation asynchronously. But this is mostly useful in shortly lived requests on web server

5.What are CPU intensive tasks?  
A.Each process is assigned a small amount of time to execute before being re-evaluated by the operating system to see if it needs to let other processes have a chance to run on the CPU. **Any process that consumes an entire time slice without pausing** would be considered CPU Intensive Sorting, search, graph traversal, matrix multiply are all CPU operations, a process is CPU-intensive or not it depends on how much and how frequent are their execution

6.How can you end up blocking your main thread in node.js?  
A. Blocking is when the execution of additional JavaScript in the Node.js process must wait until a non-JavaScript operation completes. This happens because the event loop is unable to continue running JavaScript while a blocking operation is occurring.

Synchronous methods in the Node.js standard library that use libuv are the most commonly used blocking operations. Native modules may also have blocking methods. Blocking methods execute `synchronously  
// Blocking  
const fs = require('fs');  
const data = fs.readFileSync('/file.md'); // blocks here until file is read  
console.log(data);  
moreWork(); // will run after console.log

7.What is the event loop?  
A. The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.  
Node.js is a single-threaded application, but it can support concurrency via the concept of `event` and `callbacks`. Every API of Node.js is asynchronous and being single-threaded, they use `async function calls` to maintain concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.  
\*\*Event-Driven Programming\*\*

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.

Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. The functions that listen to events act as Observers. Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through events module and EventEmitter class which are used to bind events and event-listeners

8. What are different phases in event loop?  
A. Timers - Pending callbacks - Idle, prepare - Poll - Check - close callbacks.  
- Timers : The timers phase is executed directly by the Event Loop. At the beginning of this phase the Event Loop updates its own time. Then it checks a queue, or pool, of timers. This queue consists of all timers that are currently set. The Event Loop takes the timer with the shortest wait time and compares it with the Event Loop's current time. If the wait time has elapsed, then the timer's callback is queued to be called once the call stack is empty.  
- Pending Callbacks : when your application is waiting for a file to be read, it doesn't have to necessarily wait until the system gets back to it with the content of the file. It can continue code execution and receive the file's content asynchronously when it is ready.  
- Idle / Prepare : when your application is waiting for a file to be read, it doesn't have to necessarily wait until the system gets back to it with the content of the file. It can continue code execution and receive the file's content asynchronously when it is ready.  
- Poll : The poll phase has two main functions:

1. Calculating how long it should block and poll for I/O, then
2. Processing events in the poll queue.

When the event loop enters the poll phase *and there are no timers scheduled*, one of two things will happen:

* *If the poll queue is not empty*, the event loop will iterate through its queue of callbacks executing them synchronously until either the queue has been exhausted, or the system-dependent hard limit is reached.
* *If the poll queue is empty*, one of two more things will happen:
  + If scripts have been scheduled by setImmediate(), the event loop will end the poll phase and continue to the check phase to execute those scheduled scripts.
  + If scripts have not been scheduled by setImmediate(), the event loop will wait for callbacks to be added to the queue, then execute them immediately.

Once the poll queue is empty the event loop will check for timers *whose time thresholds have been reached*. If one or more timers are ready, the event loop will wrap back to the timers phase to execute those timers' callbacks  
- Check : Node.js has a special timer, setImmediate(), and its callbacks are executed during this phase. This phase runs as soon as the poll phase becomes idle. If setImmediate() is scheduled within the I/O cycle it will always be executed before other timers regardless of how many timers are present.  
- Close callback :This phase executes the callbacks of all *close* events. For example, a close event of web socket callback, or when process.exit() is called. This is when the Event Loop is wrapping up one cycle and is ready to move to the next one. It is primarily used to clean the state of the application.

9. What is process.tick?  
A.process.nextTick() is not technically part of the event loop. Instead, the nextTickQueue will be processed after the current operation is completed, regardless of the current phase of the event loop.any time you call process.nextTick() in a given phase, all callbacks passed to process.nextTick() will be resolved before the event loop continues. This can create some bad situations because it allows you to "starve" your I/O by making recursive process.nextTick() calls, which prevents the event loop from reaching the poll phase.

1. At times it's necessary to allow a callback to run after the call stack has unwound but before the event loop continues.
2. Allow users to handle errors, cleanup any then unneeded resources, or perhaps try the request again before the event loop continues.

10. When can process.tick starve your event loop?  
A. Recursively/Repeatedly adding events to the nextTick queue using process.nextTick function can cause I/O and other queues to starve forever.

11. What is the difference between setTimeout and setInterval?  
A. setTimeout : The global setTimeout() method sets a timer which executes a function or specified piece of code once the timer expires.  
Syntax : setTimeout(func , delay)

setInterval : The setInterval() method, offered on the [Window](https://developer.mozilla.org/en-US/docs/Web/API/Window) and [Worker](https://developer.mozilla.org/en-US/docs/Web/API/Worker) interfaces, repeatedly calls a function or executes a code snippet, with a fixed time delay between each call.

This method returns an interval ID which uniquely identifies the interval, so you can remove it later by calling [clearInterval()](https://developer.mozilla.org/en-US/docs/Web/API/clearInterval).  
Syntax : setInterval(func, delay)

12.How can you make a network request with http module from the backend?  
A. Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).To include the HTTP module, use the require() method.The HTTP module creates an HTTP server that listens to server ports and gives a response back to the client.  
  
var http = require('http');  
const https = require('https');  
const options = {  
hostname: 'example.com',  
port: 443,  
path: '/todos',  
method: 'GET',  
};  
const req = https.request(options, res => { console.log(`statusCode: ${res.statusCode}`)  
res.on('data', d => {  
process.stdout.write(d);  
});  
});  
req.on('error', error => {  
console.error(error);  
});  
req.end();

13.How can you create your own events?  
A. Every action on a computer is an event. Like when a connection is made or a file is opened. Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.To include the built-in Events module use the require() method. In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:  
var events = require('events');  
var eventEmitter = new events.EventEmitter();  
You can assign event handlers to your own events with the EventEmitter object.  
In the example below we have created a function that will be executed when a "scream" event is fired.  
To fire an event, use the emit() method.  
var events = require('events');  
  
var eventEmitter = new events.EventEmitter();  
//Create an event handler:  
var myEventHandler = function () {  
 console.log('I hear a scream!');  
}  
//Assign the event handler to an event:  
eventEmitter.on('scream', myEventHandler);  
//Fire the 'scream' event:  
eventEmitter.emit('scream');

14. What are clusters?  
A. Node.js runs single threaded programming, which is very memory efficient, but to take advantage of computers multi-core systems, the Cluster module allows you to easily create child processes that each runs on their own single thread, to handle the load.

15.How does your Node.js application handle scale? Elaborate  
A.

16. What is the difference between readFile and readFileSync?  
A.The fs.readFileSync() method is an inbuilt application programming interface of fs module which is used to read the file and return its content.

In fs.readFile() method, we can read a file in a non-blocking asynchronous way, but in fs.readFileSync() method, we can read files in a synchronous way, i.e. we are telling node.js to block other parallel process and do the current file reading process.

That is, when the fs.readFileSync() method is called the original node program stops executing, and node waits for the fs.readFileSync() function to get executed, after getting the result of the method the remaining node program is executed.

syntax: fs.readFileSync( path, options )

- path: It takes the relative path of the text file. The path can be of URL type. The file can also be a file descriptor. If both the files are in the same folder just give the filename in quotes.

- options: It is an optional parameter which contains the encoding and flag, the encoding contains data specification. It’s default value is null which returns raw buffer and the flag contains indication of operations in the file. It’s default value is ‘r’.

Return Value: This method returns the content of the file.

17.What are CORS? How do you configure them? Why do you need them?  
A.Cross-Origin Resource Sharing (CORS) headers allow apps running in the browser to make requests to servers on different domains (also known as origins). CORS headers are set on the server side - the HTTP server is responsible for indicating that a given HTTP request can be cross-origin. CORS defines a way in which a browser and server can interact and determine whether or not it is safe to allow a cross-origin request.  
Configuration :   
var express = require('express')  
var cors = require('cors')  
var app = express()  
app.use(cors())  
app.get('/products/:id', function (req, res, next) {  
 res.json({msg: 'This is CORS-enabled for all origins!'})  
})  
app.listen(8080, function () {  
 console.log('CORS-enabled web server listening on port 80')  
}).  
The client and server have a different origin from each other, i.e., accessing resources from a different server. In this case, trying to make a request to a resource on the other server will fail.

This is a security concern for the browser. CORS comes into play to disable this mechanism and allow access to these resources. It will add a response header access-control-allow-origins and specify which origins are permitted. CORS ensures that we are sending the right headers.

18. What is rate limiting?  
A. Rate limiting is a technique used to control the amount of incoming or outgoing traffic within a network. In this context, network refers to the line of communication between a client (e.g., web browser) and our server (e.g., API). Thus, it is a technique that allows us to handle user requests based on some specified constraint such that:

* There is better flow of data
* There is a reduced risk of attack, i.e., improved security
* The server is never overloaded
* Users can only do as much as is allowed by the developer

19. How does middlewares work in express?  
A. Middleware functions are functions that have access to the request object (req), the response object (res), and the `next` function in the application\'s request-response cycle.  
Middleware functions can perform the following tasks:  
\* Execute any code.  
\* Make changes to the request and the response objects.  
\* End the request-response cycle.  
\* Call the next middleware in the stack.  
If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function. Otherwise, the request will be left hanging

Middleware functions that return a Promise will call `next(value)` when they reject or throw an error. `next` will be called with either the rejected value or the thrown Error.

20.What is the difference between Encryption and Hashing?  
A. Encryption is the process of converting a normal readable message known as plaintext into a garbage message or not readable message known as Ciphertext. The ciphertext obtained from the encryption can easily be transformed into plaintext using the encryption key. Some of the examples of encryption algorithms are RSA, AES, and Blowfish.  
Hashing is the process of converting the information into a key using a hash function. The original information cannot be retrieved from the hash key by any means. Generally, the hash keys are stored in the database and they are compared to check whether the original information matches or not. They are generally used to store the passwords for login. Some of the examples of a hashing algorithm are MD5, SHA256

21.What is the difference between https and http?  
A.both of these are protocols using which the information of a particular website is exchanged between Web Server and Web Browser.The difference between *http* and *https* is that *https* is much more secure compared to *http*.  
In HTTP, URL begins with “http://” whereas URL starts with “https://”

* HTTP uses port number 80 for communication and HTTPS uses 443
* HTTP is considered to be unsecure and HTTPS is secure
* HTTP Works at Application Layer and HTTPS works at Transport Layer
* In HTTP, Encryption is absent and Encryption is present in HTTPS as discussed above
* HTTP does not require any certificates and HTTPS needs SSL Certificates

22. What is TLS?  
A.Transport Layer Security (TLS), whose predecessor is Secure Sockets Layer (SSL), is the standard security technology for establishing an encrypted link between a web server and a web client, such as a browser or an app. An encrypted link ensures that all data passing between the server and the client remains private. To use TLS, a client makes a secure request to the server by using the encrypted HTTPS protocol

23.What is AES?  
A.The AES Encryption algorithm (also known as the Rijndael algorithm) is a symmetric block cipher algorithm with a block/chunk size of 128 bits. It converts these individual blocks using keys of 128, 192, and 256 bits. Once it encrypts these blocks, it joins them together to form the ciphertext.

It is based on a substitution-permutation network, also known as an SP network. It consists of a series of linked operations, including replacing inputs with specific outputs (substitutions) and others involving bit shuffling (permutations)

24. What is JWT Token? Why do we need to use JWT? What are some pros and cons?  
A. JSON Web Token (JWT) is an open standard that defines a compact and self-contained way of securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.  
Why ?   
JWTs are a good way of securely transmitting information between parties because they can be signed, which means you can be sure that the senders are who they say they are. Additionally, the structure of a JWT allows you to verify that the content hasn't been tampered with.  
Pros :  
- Protects against CSRF (Cross Site Request Forgery) attacks.  
- JWT is compact. Because of its size, it can be sent through an URL, POST parameter, or inside an HTTP header.  
- You can authorize only the requests you wish to authorize. Cookies are sent for every single request.  
- JWT is a stateless authentication mechanism as the user state is never saved in the database. As JWTs are self-contained, all the necessary information is there, reducing the need of going back and forward to the database. With JWT we don't need to query database to authenticate the user for every api call.  
Cons :   
- Not easy to revoke a JWT as it is a stateless authentication mechanism. It makes difficult to implement feature like Sign out from all devices. This is easy to implement using session based authentication as we just need to delete the session from database.  
- Need to write some code to implement whereas cookies work out of the box.

25. What is salting? Where do we store salt?  
A. Salting refers to adding random data to a hash function to obtain a unique output which refers to the hash. Even when the same input is used, it is possible to obtain different and unique hashes. These hashes aim to strengthen security, protect against dictionary attacks, brute-force attacks, and several others.  
A new salt is randomly generated for each password. Typically, the salt and the password (or its version after [key stretching](https://en.wikipedia.org/wiki/Key_stretching)) are [concatenated](https://en.wikipedia.org/wiki/Concatenation) and fed to a [cryptographic hash function](https://en.wikipedia.org/wiki/Cryptographic_hash_function), and the output [hash value](https://en.wikipedia.org/wiki/Hash_function) (but not the original password) is stored with the salt in a database. Hashing allows later [authentication](https://en.wikipedia.org/wiki/Authentication) without keeping and therefore risking exposure of the [plaintext](https://en.wikipedia.org/wiki/Plaintext) password if the authentication data store is compromised. Note that due to this, salts don't need to be encrypted or stored separately from the hashed password itself, because even if an attacker has access to the database with the hash values and the salts, the correct use of said salts will hinder common attacks.

26.What is the difference between authorisation and Authentication?  
A. - In authentication process, the identity of users are checked for providing the access to the system.While in authorization process, person’s or user’s authorities are checked for accessing the resources.  
- In authentication process, users or persons are verified.  
 While in this process, users or persons are validated.  
- It is done before the authorization process.  
 While this process is done after the authentication process.  
- It needs usually user’s login details.  
 While it needs user’s privilege or security levels  
- Authentication determines whether the person is user or not  
While Authorization determines **What permission do user have?**

**27. What is the difference between JS on the browser and node?  
A. 1.** [**NodeJS**](https://www.geeksforgeeks.org/introduction-to-nodejs/) **:   
NodeJS is a cross-platform and opensource Javascript runtime environment that allows the javascript to be run on the server-side. Nodejs allows Javascript code to run outside the browser. Nodejs comes with a lot of modules and mostly used in web development.  
2.** [**JavaScript**](https://www.geeksforgeeks.org/JavaScript-tutorial/) **:   
Javascript is a** [**Scripting language**](https://www.geeksforgeeks.org/introduction-to-scripting-languages/)**. It is mostly abbreviated as JS. It can be said that Javascript is the updated version of the ECMA script. Javascript is a high-level programming language that uses the concept of Oops but it is based on prototype inheritance.**

| **Javascript is a programming language that is used for writing scripts on the website.** | **NodeJS is a Javascript runtime environment.** |  |
| --- | --- | --- |
| **2.** | **Javascript can only be run in the browsers.** | **We can run Javascript outside the browser with the help of NodeJS.** |
| **3.** | **It is basically used on the client-side.** | **It is mostly used on the server-side.** |
| **4.** | **Javascript is capable enough to add HTML and play with the DOM.** | **Nodejs does not have capability to add HTML tags.** |
| **5.** | **Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox.** | **V8 is the Javascript engine inside of node.js that parses and runs Javascript.** |
| **6.** | **Javascript is used in frontend development.** | **Nodejs is used in server-side development.** |
| **7.** | **Some of the javascript frameworks are RamdaJS, TypedJS, etc.** | **Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm.** |
| **8.** | **It is the upgraded version of ECMA script that uses Chrome’s V8 engine written in C++.** | **Nodejs is written in C, C++ and Javascript.** |

|  |
| --- |

28.What is V8?  
A.V8 is a C++-based open-source JavaScript engine developed by Google. It was originally designed for Google Chrome and Chromium-based browsers (such as Brave) in 2008, but it was later utilized to create Node.js for server-side coding.V8 is known to be a JavaScript engine because it takes JavaScript code and executes it while browsing in Chrome. It provides a runtime environment for the execution of JavaScript code. The best part is that the JavaScript engine is completely independent of the browser in which it runs. This is the feature that led Node.js designers to choose the V8 engine to power the framework, and the rest is history. The V8 engine was also utilized to construct desktop frameworks and databases as Node.JS grew in popularity.

|  |
| --- |