INTERVIEW QUESTIONS ON NODEJS AND EXPRESSJS

NODEJS INTERVIEW QUESTIONS

1. 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.

2. 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.

3. 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.

4. 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 even-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 ample libraries so that we don’t need to reinvent the wheel.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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);
```

11. 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.

12. 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.

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

- Task Function
- Concurrency Value

14. 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")
```

15. 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.

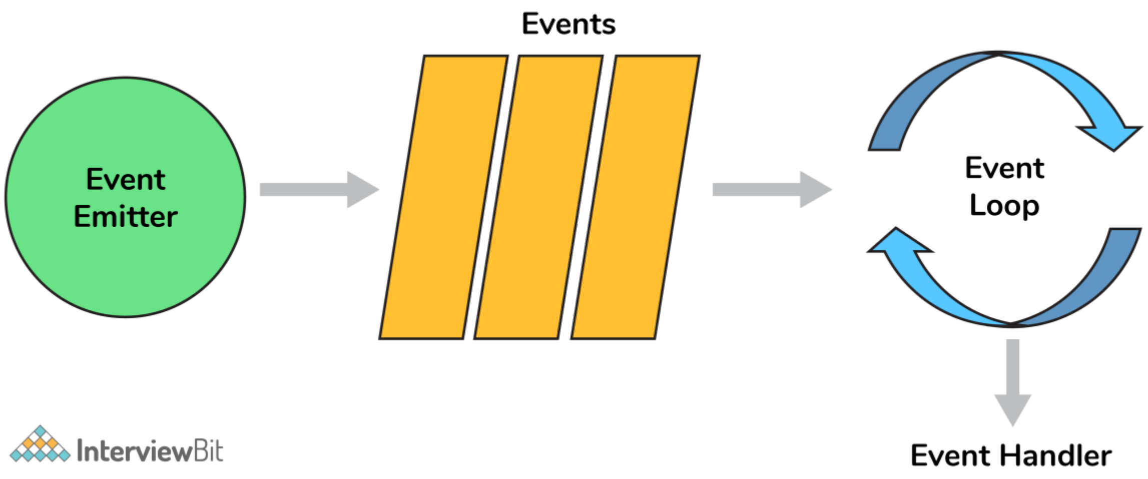
16. What do you understand by callback hell?

```
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.

17. 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.

18. 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.

19. 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.

20. 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.

21. 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);
}
 }
}
```

22. 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()).

23. 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.

24. 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.

25. 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

26. 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.

27. 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.

28. 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.

29. 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();
    });
});
});
```

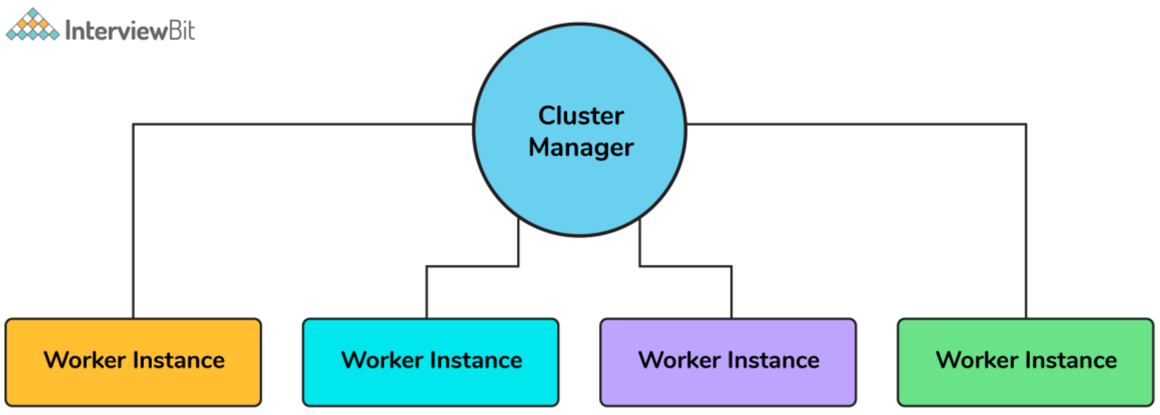
30. What is an Event Emitter in Node.js?

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.

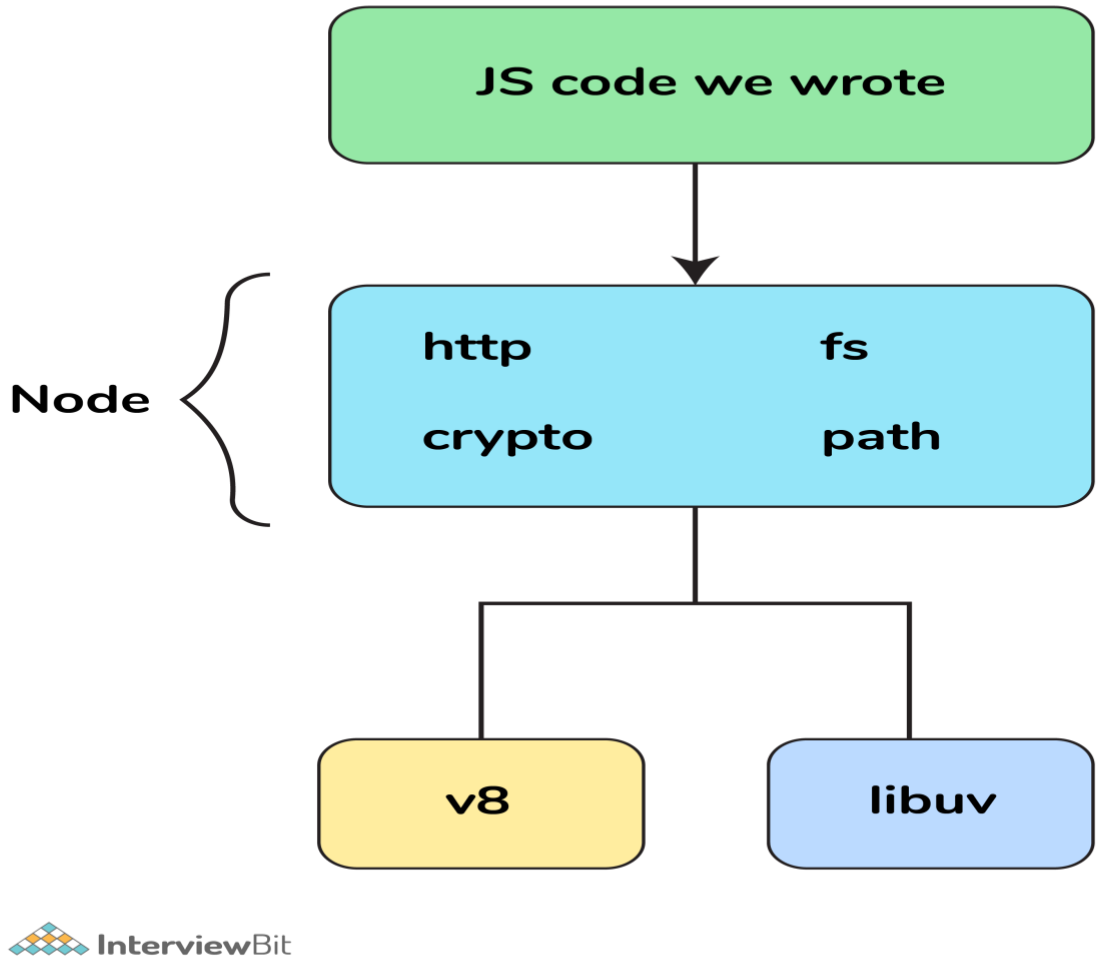
Node.js applications run on a single processor, which means that by default they don't take advantage of a multiple-core system. Cluster mode is used to start up multiple node.js processes thereby having multiple instances of the event loop. When we start using cluster in a nodejs app behind the scene 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

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 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');
});
```

1. What is Node.js? Where can you use it?

Node.js is an open-source, cross-platform JavaScript runtime environment and library to run web applications outside the client's browser. It is used to create server-side web applications.

Node.js is perfect for data-intensive applications as it uses an asynchronous, event-driven model. You can use I/O intensive web applications like video streaming sites. You can also use it for developing: Real-time web applications, Network applications, General-purpose applications, and Distributed systems.

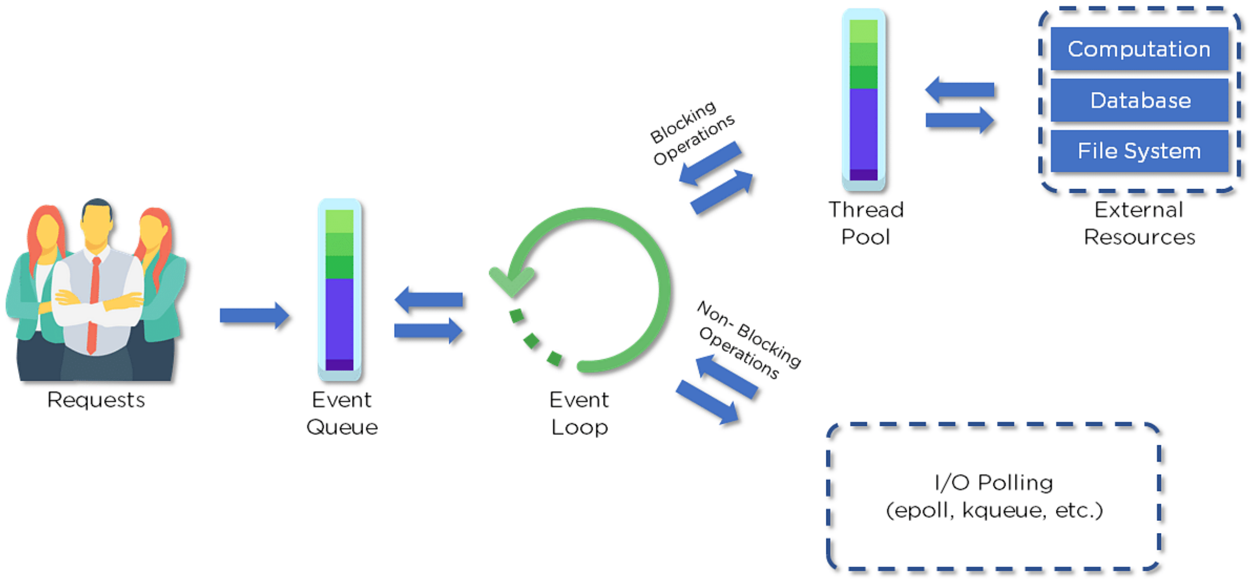
2. Why use Node.js?

Node.js makes building scalable network programs easy. Some of its advantages include:

- It is generally fast
- It rarely blocks
- It offers a unified programming language and data type
- Everything is asynchronous
- It yields great concurrency

3. How does Node.js work?

A web server using Node.js typically has a workflow that is quite similar to the diagram illustrated below. Let's explore this flow of operations in detail.



- Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking:
- Querying for data
- Deleting data
- Updating the data
- Node.js retrieves the incoming requests and adds those to the Event Queue
- The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough not to require any external resources
- The Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients

A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.

Once the task is carried out completely, the response is sent to the Event Loop that sends that response back to the client.

4. Why is Node.js Single-threaded?

Node.js is single-threaded for async processing. By doing async processing on a single-thread under typical web loads, more performance and scalability can be achieved instead of the typical thread-based implementation.

5. If Node.js is single-threaded, then how does it handle concurrency?

The Multi-Threaded Request/Response Stateless Model is not followed by the Node JS Platform, and it adheres to the Single-Threaded Event Loop Model. The Node JS Processing paradigm is heavily influenced by the JavaScript Event-based model and the JavaScript callback system. As a result, Node.js can easily manage more concurrent client requests. The event loop is the processing model's beating heart in Node.js.

6. Explain callback in Node.js.

A callback function is called after a given task. It allows other code to be run in the meantime and prevents any blocking. Being an asynchronous platform, Node.js heavily relies on callback. All APIs of Node are written to support callbacks.

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

- The control flow of asynchronous logic is more specified and structured.
- The coupling is low.
- We've built-in error handling.
- Improved readability.

8. How would you define the term I/O?

- The term I/O is used to describe any program, operation, or device that transfers data to or from a medium and to or from another medium
- Every transfer is an output from one medium and an input into another. The medium can be a physical device, network, or files within a system

9. How is Node.js most frequently used?

Node.js is widely used in the following applications:

1. Real-time chats
2. Internet of Things
3. Complex SPAs (Single-Page Applications)

4. Real-time collaboration tools
5. Streaming applications
6. Microservices architecture

10. Explain the difference between frontend and backend development?

Front-end	Back-end
Frontend refers to the client-side of an application	Backend refers to the server-side of an application
It is the part of a web application that users can see and interact with	It constitutes everything that happens behind the scenes
It typically includes everything that attributes to the visual aspects of a web application	It generally includes a web server that communicates with a database to serve requests
HTML, CSS, JavaScript, AngularJS, and ReactJS are some of the essentials of frontend development	Java, PHP, Python, and Node.js are some of the backend development technologies

11. What is NPM?

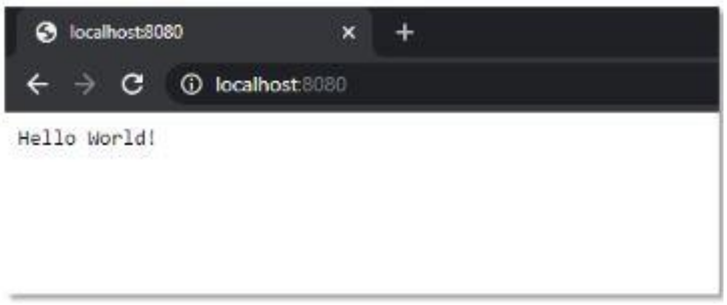
NPM stands for Node Package Manager, responsible for managing all the packages and modules for Node.js.

Node Package Manager provides two main functionalities:

- Provides online repositories for node.js packages/modules, which are searchable on search.nodejs.org
- Provides command-line utility to install Node.js packages and also manages Node.js versions and dependencies

12. What are the modules in Node.js?

Modules are like JavaScript libraries that can be used in a Node.js application to include a set of functions. To include a module in a Node.js application, use the require() function with the parentheses containing the module's name.



```
Hello World!
```

Node.js has many modules to provide the basic functionality needed for a web application. Some of them include:

Core Modules	Description
HTTP	Includes classes, methods, and events to create a Node.js HTTP server
util	Includes utility functions useful for developers
fs	Includes events, classes, and methods to deal with file I/O operations
url	Includes methods for URL parsing
query string	Includes methods to work with query string
stream	Includes methods to handle streaming data
zlib	Includes methods to compress or decompress files

13. What is the purpose of the module .Exports?

In Node.js, a module encapsulates all related codes into a single unit of code that can be parsed by moving all relevant functions into a single file. You may export a module with the module and export the function, which lets it be imported into another file with a needed keyword.

14. Why is Node.js preferred over other backend technologies like Java and PHP?

Some of the reasons why Node.js is preferred include:

- Node.js is very fast
- Node Package Manager has over 50,000 bundles available at the developer's disposal
- Perfect for data-intensive, real-time web applications, as Node.js never waits for an API to return data
- Better synchronization of code between server and client due to same code base
- Easy for web developers to start using Node.js in their projects as it is a JavaScript library

15. What is the difference between Angular and Node.js?

Angular	Node.js
It is a frontend development framework	It is a server-side environment
It is written in TypeScript	It is written in C, C++ languages
Used for building single-page, client-side web applications	Used for building fast and scalable server-side networking applications
Splits a web application into MVC components	Generates database queries

16. Which database is more popularly used with Node.js?

MongoDB is the most common database used with Node.js. It is a NoSQL, cross-platform, document-oriented database that provides high performance, high availability, and easy scalability.

17. What are some of the most commonly used libraries in Node.js?

There are two commonly used libraries in Node.js:

- ExpressJS - Express is a flexible Node.js web application framework that provides a wide set of features to develop web and mobile applications.
- Mongoose - Mongoose is also a Node.js web application framework that makes it easy to connect an application to a database.

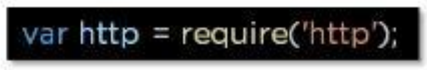
18. What are the pros and cons of Node.js?

Node.js Pros	Node.js Cons
Fast processing and an event-based model	Not suitable for heavy computational tasks
Uses JavaScript, which is well-known amongst developers	Using callback is complex since you end up with several nested callbacks
Node Package Manager has over 50,000 packages that provide the functionality to an application	Dealing with relational databases is not a good option for Node.js
Best suited for streaming huge amounts of data and I/O intensive operations	Since Node.js is single-threaded, CPU intensive tasks are not its strong suit

19. What is the command used to import external libraries?

The “require” command is used for importing external libraries. For example - “var http=require (“HTTP”).” This will load the HTTP library and the single exported object through the HTTP variable.

Now that we have covered some of the important beginner-level Node.js interview questions let us look at some of the intermediate-level Node.js interview questions.



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

20. What does event-driven programming mean?

An event-driven programming approach uses events to trigger various functions. An event can be anything, such as typing a key or clicking a mouse button. A call-back function is already registered with the element executes whenever an event is triggered.

21. What is an Event Loop in Node.js?

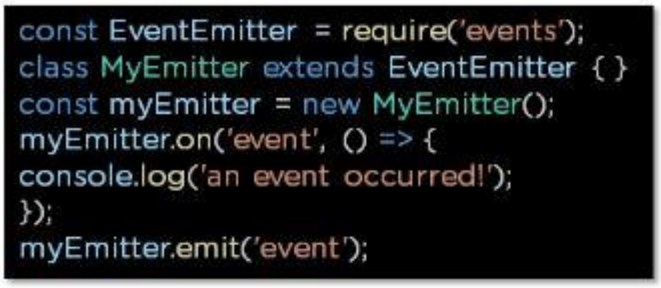
Event loops handle asynchronous callbacks in Node.js. It is the foundation of the non-blocking input/output in Node.js, making it one of the most important environmental features.

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

The distinction between method and product. This is accomplished through the use of nextTick() and setImmediate(). next Tick() postpones the execution of action until the next pass around the event loop, or it simply calls the callback function once the event loop's current execution is complete, whereas setImmediate() executes a callback on the next cycle of the event loop and returns control to the event loop for any I/O operations.

23. What is an EventEmitter in Node.js?

- EventEmitter is a class that holds all the objects that can emit events
- Whenever an object from the EventEmitter class throws an event, all attached functions are called upon synchronously



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

24. What are the two types of API functions in Node.js?

The two types of API functions in Node.js are:

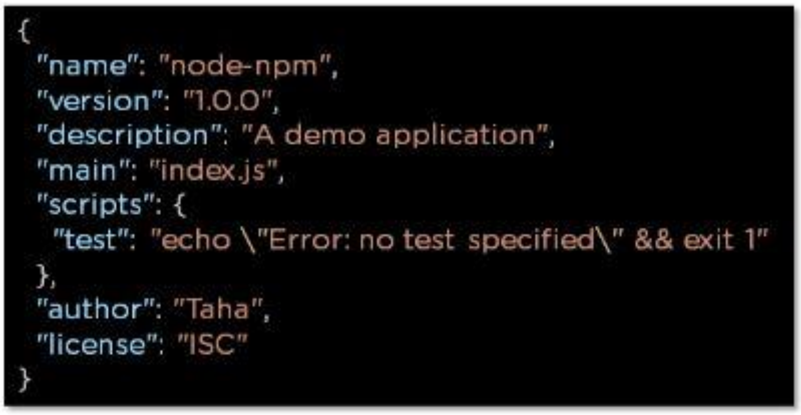
- Asynchronous, non-blocking functions
- Synchronous, blocking functions

25. What is the package.json file?

The package.json file is the heart of a Node.js system. This file holds the metadata for a particular project. The package.json file is found in the root directory of any Node application or module

This is what a package.json file looks like immediately after creating a Node.js project using the command: npm init

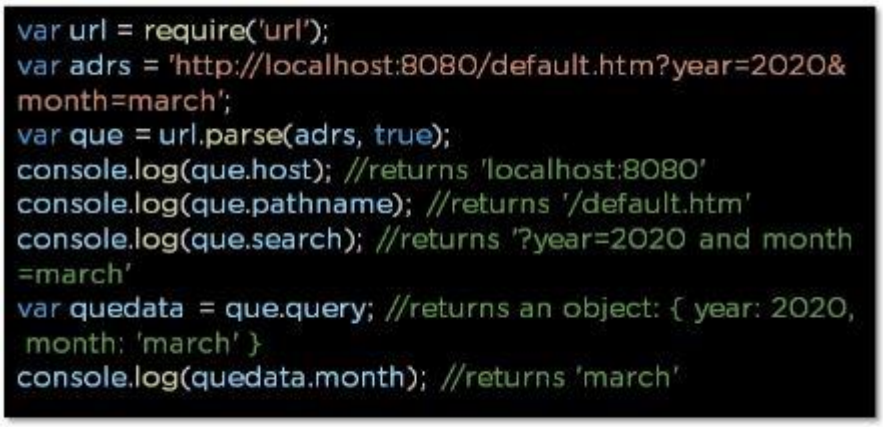
You can edit the parameters when you create a Node.js project.



```
{
  "name": "node-npm",
  "version": "1.0.0",
  "description": "A demo application",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Taha",
  "license": "ISC"
}
```

26. How would you use a URL module in Node.js?

The URL module in Node.js provides various utilities for URL resolution and parsing. It is a built-in module that helps split up the web address into a readable format.



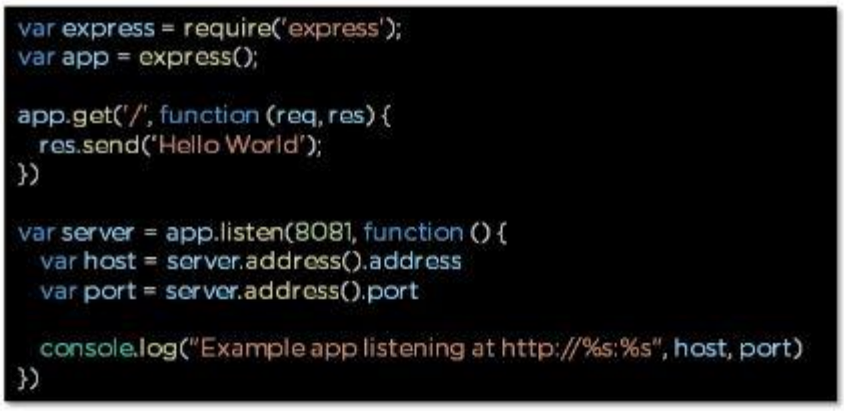
```
var url = require('url');
var adrs = 'http://localhost:8080/default.htm?year=2020&
month=march';
var que = url.parse(adrs, true);
console.log(que.host); //returns 'localhost:8080'
console.log(que.pathname); //returns '/default.htm'
console.log(que.search); //returns '?year=2020 and month
=march'
var quedata = que.query; //returns an object: { year: 2020,
 month: 'march' }
console.log(quedata.month); //returns 'march'
```

27. What is the Express.js package?

Express is a flexible Node.js web application framework that provides a wide set of features to develop both web and mobile applications

28. How do you create a simple Express.js application?

- The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on
- The response object represents the HTTP response that an Express app sends when it receives an HTTP request



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

app.get('/', function (req, res) {
  res.send('Hello World');
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})
```

29. What are streams in Node.js?

Streams are objects that enable you to read data or write data continuously.

There are four types of streams:

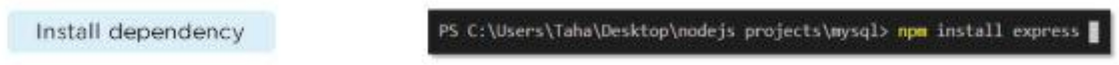
Readable – Used for reading operations

Writable – Used for write operations

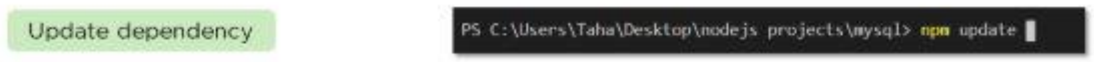
Duplex – Can be used for both reading and write operations

Transform – A type of duplex stream where the output is computed based on input

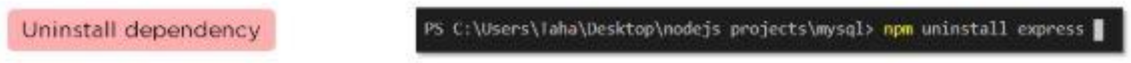
30. How do you install, update, and delete a dependency?


Install dependency

```
PS C:\Users\Taha\Desktop\nodejs projects\mysql> npm install express
```

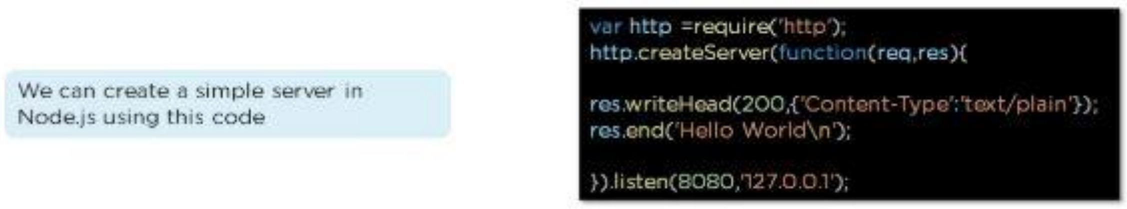

Update dependency

```
PS C:\Users\Taha\Desktop\nodejs projects\mysql> npm update
```


Uninstall dependency

```
PS C:\Users\Taha\Desktop\nodejs projects\mysql> npm uninstall express
```

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


We can create a simple server in Node.js using this code

```
var http =require('http');
http.createServer(function(req,res){

res.writeHead(200,{'Content-Type':'text/plain'});
res.end('Hello World\n');

}).listen(8080,'127.0.0.1');
```

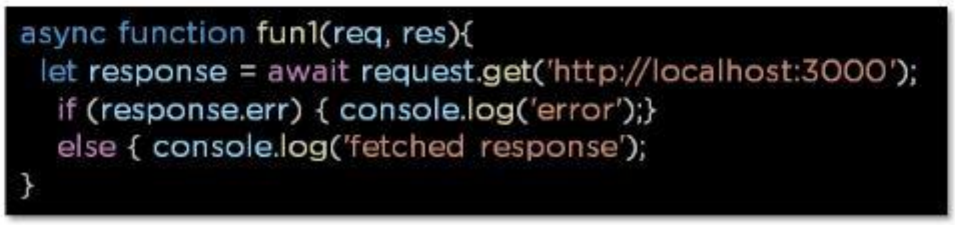
- Import the HTTP module
- Use createServer function with a callback function using request and response as parameters.
- Type “hello world."
- Set the server to listen to port 8080 and assign an IP address

32. Explain asynchronous and non-blocking APIs in Node.js.

- All Node.js library APIs are asynchronous, which means they are also non-blocking
- A Node.js-based server never waits for an API to return data. Instead, it moves to the next API after calling it, and a notification mechanism from a Node.js event responds to the server for the previous API call

33. How do we implement async in Node.js?

As shown below, the async code asks the JavaScript engine running the code to wait for the request.get() function to complete before moving on to the next line for execution.



```
async function fun1(req, res){
  let response = await request.get('http://localhost:3000');
   if (response.err) { console.log('error');}
   else { console.log('fetched response');
}
```

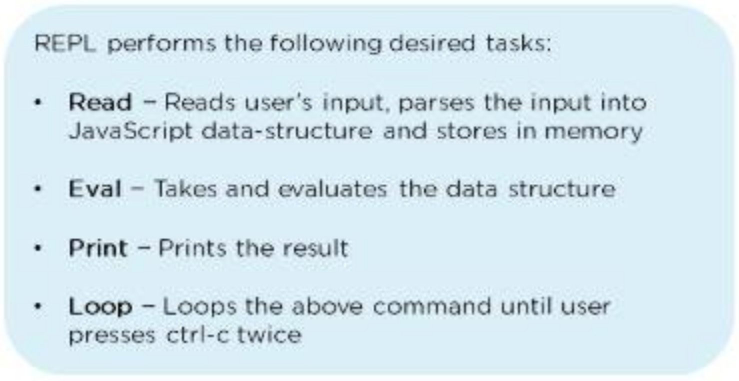
34. What is a callback function in Node.js?

A callback is a function called after a given task. This prevents any blocking and enables other code to run in the meantime.

In the last section, we will now cover some of the advanced-level Node.js interview questions.

35. What is REPL in Node.js?

REPL stands for Read Eval Print Loop, and it represents a computer environment. It’s similar to a Windows console or Unix/Linux shell in which a command is entered. Then, the system responds with an output



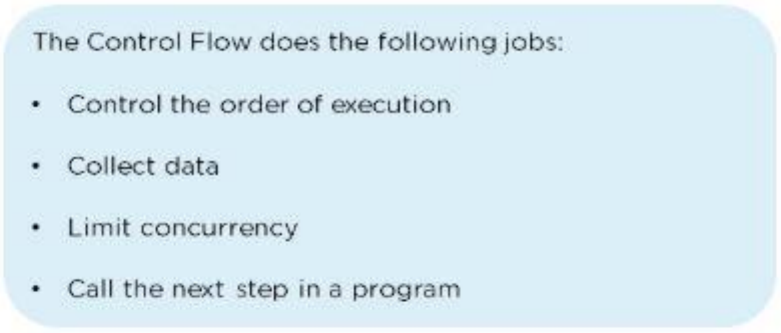
REPL performs the following desired tasks:

- **Read** – Reads user's input, parses the input into JavaScript data-structure and stores in memory
- **Eval** – Takes and evaluates the data structure
- **Print** – Prints the result
- **Loop** – Loops the above command until user presses ctrl-c twice

36. What is the control flow function?

The control flow function is a piece of code that runs in between several asynchronous function calls.

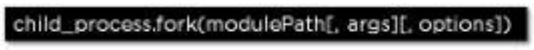
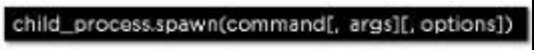
37. How does control flow manage the function calls?



The Control Flow does the following jobs:

- Control the order of execution
- Collect data
- Limit concurrency
- Call the next step in a program

38. What is the difference between fork() and spawn() methods in Node.js?

fork()	spawn()
 `child_process.fork(modulePath[, args][, options])`	 `child_process.spawn(command[, args][, options])`
fork() is a particular case of spawn() that generates a new instance of a V8 engine.	Spawn() launches a new process with the available set of commands.
Multiple workers run on a single node code base for multiple tasks.	This method doesn't generate a new V8 instance, and only a single copy of the node module is active on the processor.

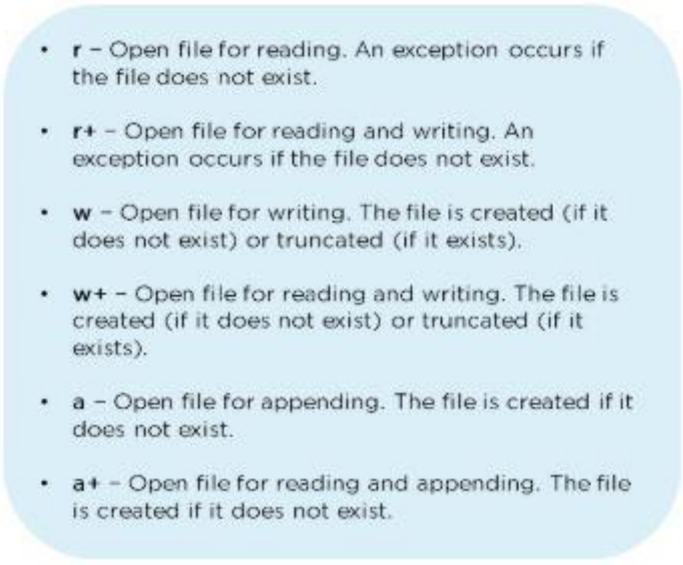
39. What is the buffer class in Node.js?

Buffer class stores raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap. Buffer class is used because pure JavaScript is not compatible with binary data

40. What is piping in Node.js?

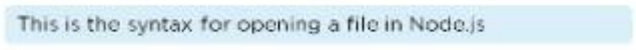
Piping is a mechanism used to connect the output of one stream to another stream. It is normally used to retrieve data from one stream and pass output to another stream

41. What are some of the flags used in the read/write operations in files?

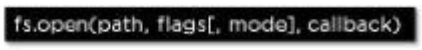


- **r** – Open file for reading. An exception occurs if the file does not exist.
- **r+** – Open file for reading and writing. An exception occurs if the file does not exist.
- **w** – Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
- **w+** – Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
- **a** – Open file for appending. The file is created if it does not exist.
- **a+** – Open file for reading and appending. The file is created if it does not exist.

42. How do you open a file in Node.js?



This is the syntax for opening a file in Node.js



```
fs.open(path, flags[, mode], callback)
```

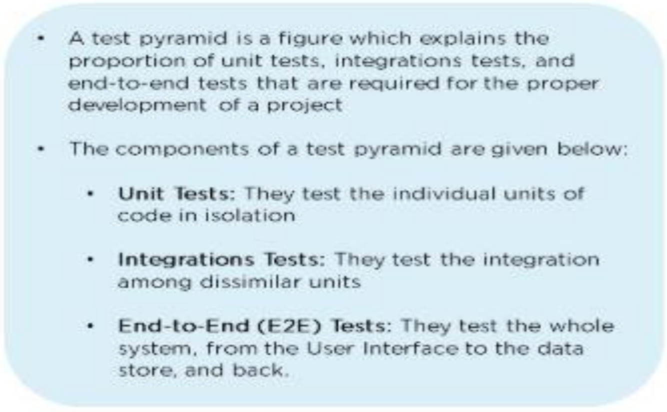
43. What is callback hell?

- Callback hell, also known as the pyramid of doom, is the result of intensively nested, unreadable, and unmanageable callbacks, which in turn makes the code harder to read and debug
- improper implementation of the asynchronous logic causes callback hell

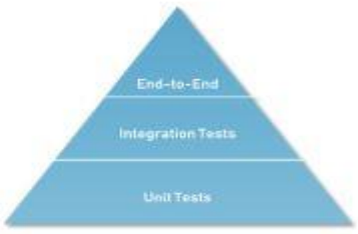
44. What is a reactor pattern in Node.js?

A reactor pattern is a concept of non-blocking I/O operations. This pattern provides a handler that is associated with each I/O operation. As soon as an I/O request is generated, it is then submitted to a demultiplexer

45. What is a test pyramid in Node.js?



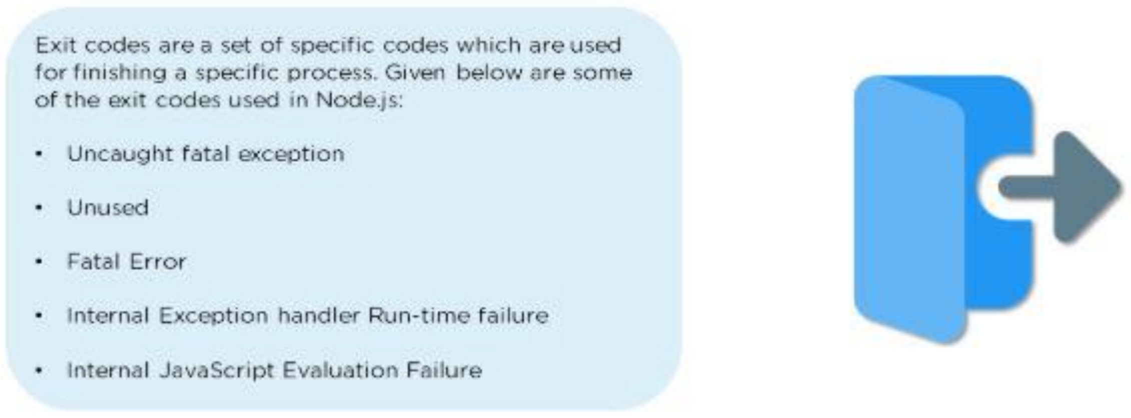
- A test pyramid is a figure which explains the proportion of unit tests, integrations tests, and end-to-end tests that are required for the proper development of a project
- The components of a test pyramid are given below:
  - **Unit Tests:** They test the individual units of code in isolation
  - **Integrations Tests:** They test the integration among dissimilar units
  - **End-to-End (E2E) Tests:** They test the whole system, from the User Interface to the data store, and back.



46. For Node.js, why does Google use the V8 engine?

The V8 engine, developed by Google, is open-source and written in C++. Google Chrome makes use of this engine. V8, unlike the other engines, is also utilized for the popular Node.js runtime. V8 was initially intended to improve the speed of JavaScript execution within web browsers. Instead of employing an interpreter, V8 converts JavaScript code into more efficient machine code to increase performance. It turns JavaScript code into machine code during execution by utilizing a JIT (Just-In-Time) compiler, as do many current JavaScript engines such as SpiderMonkey or Rhino (Mozilla).

47. Describe Node.js exit codes.



Exit codes are a set of specific codes which are used for finishing a specific process. Given below are some of the exit codes used in Node.js:

- Uncaught fatal exception
- Unused
- Fatal Error
- Internal Exception handler Run-time failure
- Internal JavaScript Evaluation Failure

48. Explain the concept of middleware in Node.js.

Middleware is a function that receives the request and response objects. Most tasks that the middleware functions perform are:

- Execute any code
- Update or modify the request and the response objects
- Finish the request-response cycle
- Invoke the next middleware in the stack

49. What are the different types of HTTP requests?

HTTP defines a set of request methods used to perform desired actions. The request methods include:

GET: Used to retrieve the data

POST: Generally used to make a change in state or reactions on the server

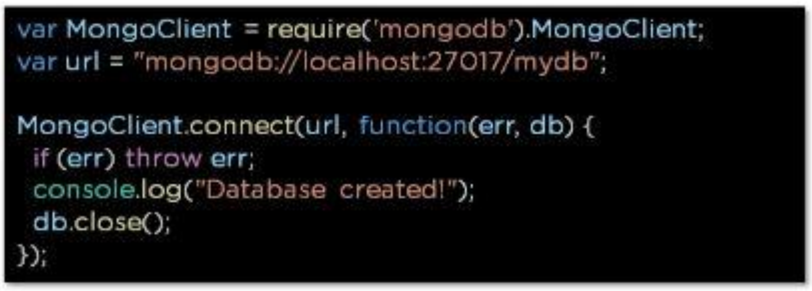
HEAD: Similar to the GET method, but asks for the response without the response body

DELETE: Used to delete the predetermined resource

50. How would you connect a MongoDB database to Node.js?

To create a database in MongoDB:

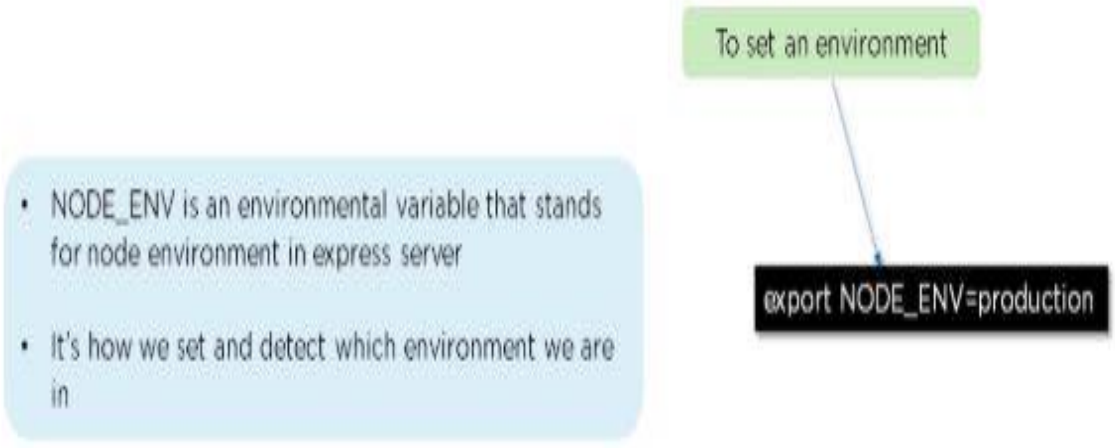
- Start by creating a MongoClient object
- Specify a connection URL with the correct IP address and the name of the database you want to create



```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

51. What is the purpose of NODE_ENV?

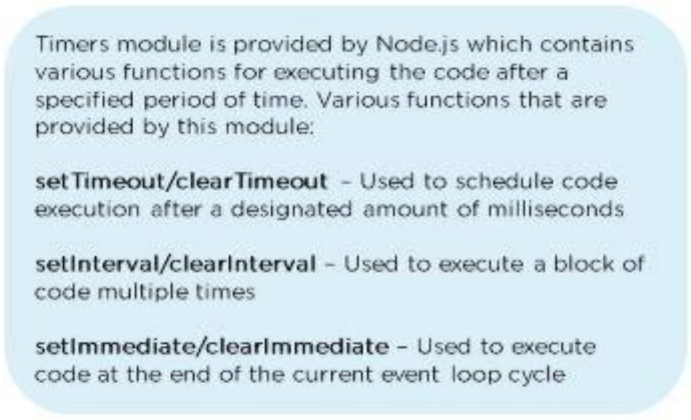


- NODE_ENV is an environmental variable that stands for node environment in express server
- It's how we set and detect which environment we are in

```
export NODE_ENV=production
```

52. List the various Node.js timing features.

As you prepare for your upcoming job interview, we hope that this comprehensive guide has provided more insight into what types of questions you’ll be asked.



Timers module is provided by Node.js which contains various functions for executing the code after a specified period of time. Various functions that are provided by this module:

setTimeout/clearTimeout - Used to schedule code execution after a designated amount of milliseconds

setInterval/clearInterval - Used to execute a block of code multiple times

setImmediate/clearImmediate - Used to execute code at the end of the current event loop cycle

53. What is WASI, and why is it being introduced?

The WASI class implements the WASI system called API and extra convenience methods for interacting with WASI-based applications. Every WASI instance represents a unique sandbox environment. Each WASI instance must specify its command-line parameters, environment variables, and sandbox directory structure for security reasons.

1) What is Node.js?

Node.js is Server-side scripting which is used to build scalable programs. It is a web application framework built on Google Chrome's JavaScript Engine. It runs within the Node.js runtime on Mac OS, Windows, and Linux with no changes. This runtime facilitates you to execute a JavaScript code on any machine outside a browser.

2) Is Node.js free to use?

Yes. It is released under MIT license and is free to use.

3) Is Node a single threaded application?

Yes. Node is a single-threaded application with event looping.

4) What is the purpose of Node.js?

These are the following purposes of Node.js:

- Real-time web applications
- Network applications
- Distributed systems
- General purpose applications

5) What are the advantages of Node.js?

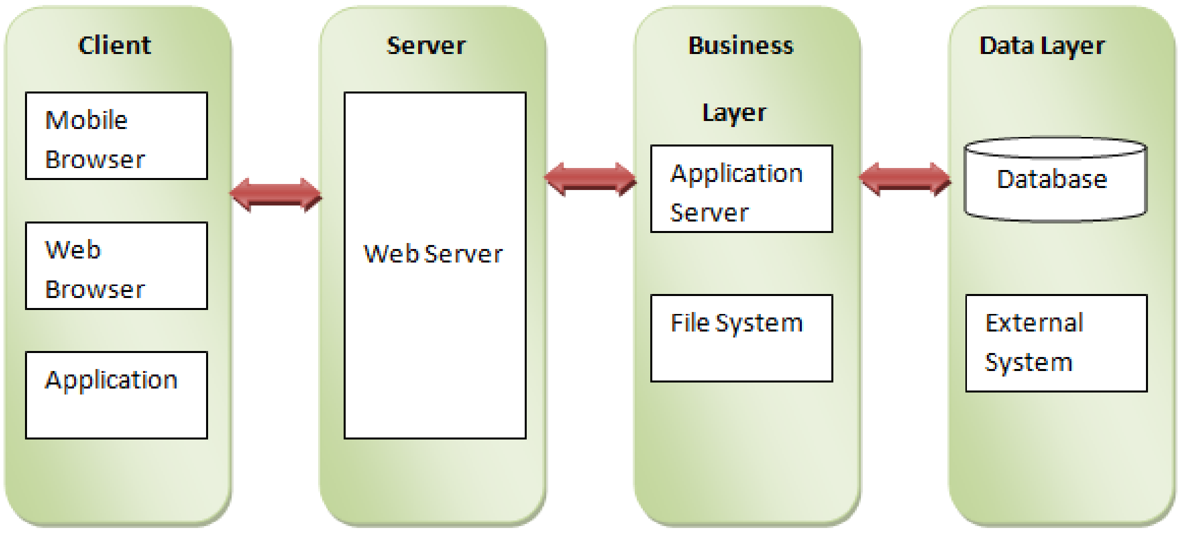
Following are the main advantages of Node.js:

- Node.js is asynchronous and event-driven. All API?s of Node.js library are non-blocking, and its server doesn't wait for an API to return data. It moves to the next API after calling it, and a notification mechanism of Events of Node.js responds to the server from the previous API call.
- Node.js is very fast because it builds on Google Chrome?s V8 JavaScript engine. Its library is very fast in code execution.
- Node.js is single threaded but highly scalable.
- Node.js provides a facility of no buffering. Its application never buffers any data. It outputs the data in chunks.

6) Explain Node.js web application architecture?

A web application distinguishes into 4 layers:

- **Client Layer:** The Client layer contains web browsers, mobile browsers or applications which can make an HTTP request to the web server.
- **Server Layer:** The Server layer contains the Web server which can intercept the request made by clients and pass them the response.
- **Business Layer:** The business layer contains application server which is utilized by the web server to do required processing. This layer interacts with the data layer via database or some external programs.
- **Data Layer:** The Data layer contains databases or any source of data.



7) What do you understand by the term I/O?

The term I/O stands for input and output. It is used to access anything outside of your application. The I/O describes any program, operation, or device that transfers data to or from a medium or another medium. This medium can be a physical device, network, or files within a system.

I/O is loaded into the machine memory to run the program once the application starts.

8) How many types of API functions are available in Node.js?

There are two types of API functions in Node.js:

- Asynchronous, Non-blocking functions
- Synchronous, Blocking functions

9) What do you understand by the first class function in JavaScript?

When functions are treated like any other variable, then those functions are called first-class functions. Apart from JavaScript, many other programming languages, such as Scala, Haskell, etc. follow this pattern. The first class functions can be passed as a param to another function (callback), or a function can return another function (higher-order function). Some examples of higher-order functions that are popularly used are map() and filter().

10) What is the difference between JavaScript and Node.js?

Difference between JavaScript and Node.js

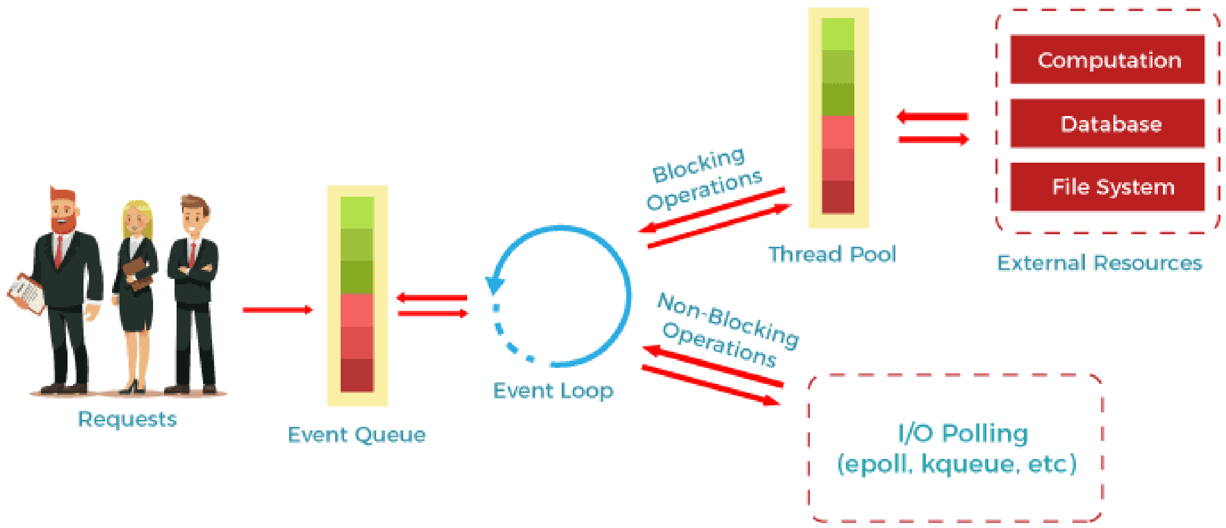
The following table specifies the crucial differences between JavaScript and Node.js:

Comparison features	**JavaScript**	**Node.js**
Type	JavaScript is a programming language. More precisely, you can say that it is a scripting language used for writing scripts on the website.	Node.js is an interpreter and run time environment for JavaScript.
Utility	JavaScript is used for any client-side activity for a web application.	Node.js is used for accessing or performing any non-blocking operation of any operating system.
Running Engine	The running engine for JavaScript is Spider monkey (Firefox),	The running engine for Node.js is V8 (Google Chrome).

	JavaScript Core (Safari), V8 (Google Chrome), etc.	
Browser compatibility	JavaScript can only be run in browsers.	The Node.js code can be run outside the browser.
Platform dependency	JavaScript is basically used on the client-side and is used in frontend development.	Node.js is mostly used on the server-side and is used in server-side development.
HTML compatibility	JavaScript is capable enough to add HTML and play with the DOM.	Node.js is not compatible enough to add HTML tags.
Examples	Some examples of the JavaScript frameworks are RamdaJS, TypedJS, etc.	Some examples of the Node.js modules are Lodash, express, etc. We have to import these modules from npm.
Written in	JavaScript is the upgraded version of ECMA script that uses Chrome's V8 engine and is written in C++.	Node.js is written in C, C++, and Javascript.

11) Explain the working of Node.js?

The workflow of a Node.js web server typically looks like the following diagram. Let us see the flow of operations in detail:



- According to the above diagram, the clients send requests to the webserver to interact with the web application. These requests can be non-blocking or blocking and used for querying the data, deleting data, or updating the data.
- js receives the incoming requests and adds those to the Event Queue.
- After this step, the requests are passed one by one through the Event Loop. It checks if the requests are simple enough not to require any external resources.
- The event loop then processes the simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients.
- A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.
- Once the task is completed, the response is sent to the Event Loop that sends that response back to the client.

12) How can you manage the packages in your Node.js project?

We can manage the packages in our Node.js project by using several package installers and their configuration file accordingly. Most of them use npm or yarn. The npm and yarn both provide almost all libraries of JavaScript with extended features of controlling environment-specific configurations. We can use package.json and package-lock.json to maintain versions of libs being installed in a project. So, there is no issue in porting that app to a different environment.

13) Why is Node.js Single-threaded?

Node.js is a single-threaded application with event looping for async processing. The biggest advantage of doing async processing on a single thread under typical web loads is that you can achieve more performance and scalability than the typical thread-based implementation.

14) What do you understand by callback hell in Node.js?

Callback hell is a phenomenon that creates a lot of problems for a JavaScript developer when he tries to execute multiple asynchronous operations one after the other. A function is called an asynchronous function when some external activity must complete before processing a result. It is called asynchronous because there is an unpredictable amount of time before a result becomes available. These functions require a callback function to handle errors and process the result.

Example:

```
getData(function(a){
    getMoreData(a, function(b){
        getMoreData(b, function(c){
            getMoreData(c, function(d){
                getMoreData(d, function(e){
                    ...
                });
            });
        });
    });
});
```

15) How is Node.js better than other most popular frameworks?

Based on the following criteria, we can say that Node.js is better than other most popular frameworks:

- js makes development simple because of its non-blocking I/O and even-based model. This simplicity results in short response time and concurrent processing, unlike other frameworks where developers use thread management.
- js runs on a chrome V8 engine which is written in C++. It enhances its performance highly with constant improvement.

- With Node.js, we will use JavaScript in both the frontend and backend development that will be much faster.
- js provides ample libraries so that we don't need to reinvent the wheel.

16) In which types of applications is Node.js most frequently used?

Node.js is most frequently and widely used in the following applications:

- Internet of Things
- Real-time collaboration tools
- Real-time chats
- Complex SPAs (Single-Page Applications)
- Streaming applications
- Microservices architecture etc.

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

Following is a list of some commonly used timing features of Node.js:

- **setTimeout/clearTimeout:** This timing feature of Node.js is used to implement delays in the code execution.
- **setInterval/clearInterval:** The setInterval or clearInterval timing feature is used to run a code block multiple times in the application.
- **setImmediate/clearImmediate:** This timing feature of Node.js is used to set the execution of the code at the end of the event loop cycle.
- **nextTick:** This timing feature sets the execution of code at the beginning of the next event loop cycle.

18) What do you understand by the term fork in Node.js?

Generally, a fork is used to spawn child processes. In Node.js, it is used to create a new **instance of the V8 engine to run multiple workers to execute the code.**

19) Which is the best tool we can use to assure consistent code style in Node.js?

ESLint tool is one of the best tools we can use with any IDE to ensure a consistent coding style. It also helps in maintaining the codebase.

20) What is the main difference between front-end and back-end development?

The following table specifies the key differences between a front-end and back-end development:

Front-end Development	Back-end Development
The front-end development in an application refers to the client-side of an application.	The back-end development in an application refers to the server-side of an application.
As the name specifies, the front-end development is the part of a web application where users can see and interact.	As the name specifies, the back-end development consists of everything that happens behind the scenes and users cannot see and interact with.
The front-end development includes everything that attributes to the visual aspects of a web application.	The back-end development generally includes a web server that communicates with the database to serve the users' requests.
HTML, CSS, Bootstrap, jQuery, JavaScript, AngularJS, and React.js are essential front-end development technologies.	Java, PHP, Python, C++, Node.js, etc., are the technologies required for back-end development.
Examples of some front-end frameworks are AngularJS, React.js, jQuery, Sass, etc.	Examples of some back-end frameworks are Express, Django, Rails, Laravel, Spring, etc.

21) Give an example to demonstrate how can we use async await in Node.js?

See the following example of using async-await pattern:

```
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);
}
 }
}
```

22) What are the modules in Node.js? Which are the different modules used in Node.js?

In Node.js applications, modules are like JavaScript libraries and include a set of functions. To include a module in a Node.js application, we must use the require() function with the parentheses containing the module's name.

Node.js has several modules which are used to provide the basic functionality needed for a web application. Following is a list of some of them:

Core Modules	Description
HTTP:	The HTTP module includes classes, methods, and events to create a Node.js HTTP server.
util:	The util module includes utility functions required in the application and is very useful for developers.
url:	The url module is used to include the methods for URL parsing.
fs:	The fs module includes events, classes, and methods to handle the

	file I/O operations.
stream:	The stream module is used to include the methods to handle streaming data.
query string:	The query string module is used to include the methods to work with a query string.
zlib:	The zlib module is used to include the methods to compress or decompress the files used in an application.

23) What are buffers in Node.js?

In general, a buffer is a temporary memory mainly used by the stream to hold on to some data until it is consumed. Buffers are used to represent a fixed-size chunk of memory allocated outside of the V8 JavaScript engine. It can't be resized. It is like an array of integers, which each represents a byte of data. It is implemented by the Node. js Buffer class. Buffers also support legacy encodings like ASCII, utf-8, etc.

24) What is error-first callback?

Error-first callbacks are used to pass errors and data. If something goes wrong, the programmer has to check the first argument because it is always an error argument. Additional arguments are used to pass data.

```
fs.readFile(filePath, function(err, data) {
  if (err) {
    //handle the error
  }
  // use the data object
});
```

25) What is an asynchronous API?

All the API's of Node.js library are asynchronous means non-blocking. A Node.js based server never waits for an API to return data. The Node.js server moves to the next API after calling it, and a notification mechanism of Events of Node.js responds to the server for the previous API call.

26) How can you avoid callbacks?

To avoid callbacks, you can use any one of the following options:

- You can use **modularization**. It breaks callbacks into independent functions.

- You can use **promises**.
- You can use **yield** with Generators and Promises.

27) Does Node.js provide Debugger?

Yes, Node.js provides a simple TCP based protocol and built-in debugging client. For debugging your JavaScript file, you can use debug argument followed by the js file name you want to debug.

Syntax:

```
node debug [script.js | -e "script" | <host>:<port>]
```

28) What is a control flow function?

Control flow function is a generic piece of code that runs in between several asynchronous function calls.

29) How "Control Flow" controls the functions calls?

The control flow does the following job:

- Control the order of execution
- Collect data
- Limit concurrency
- Call the next step in a program

30) Is it possible to access DOM in Node?

No, it is not possible to access DOM in Node.

31) What types of tasks can be done asynchronously using the event loop?

- I/O operations
- Heavy computation
- Anything requiring blocking

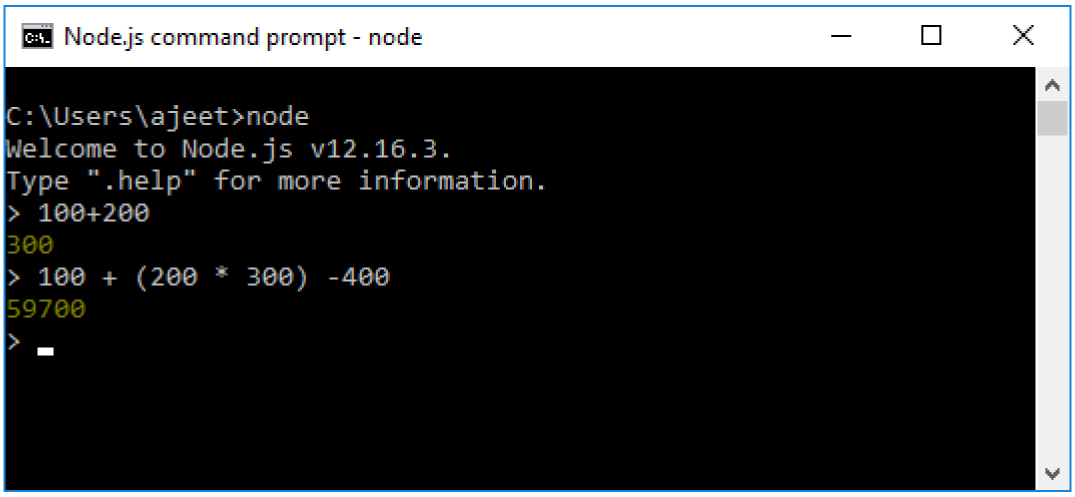
32) What is REPL in Node.js?

REPL stands for Read Eval Print Loop. It specifies a computer environment like a window console or Unix/Linux shell where you can enter a command, and the computer responds with an output. It is very useful in writing and debugging the codes. REPL environment incorporates Node.js.

See the Example:

```
$ node
> 100 + 200
300
```

4. > 100 + (200 * 300) - 400
5. 59700
6. >



Node.js command prompt - node

```
C:\Users\ajeet>node
Welcome to Node.js v12.16.3.
Type ".help" for more information.
> 100+200
300
> 100 + (200 * 300) -400
59700
>
```

33) Explain the tasks of terms used in Node REPL.

Following are the terms used in REPL with their defined tasks:

Read: It reads user's input; parse the input into JavaScript data-structure and stores in memory.

Eval: It takes and evaluates the data structure.

Print: It is used to print the result.

Loop: It loops the above command until user press ctrl-c twice to terminate.

34) Is it possible to evaluate simple expressions using Node REPL?

Yes. You can evaluate simple expressions using Node REPL.

35) What is the use of the underscore variable in REPL?

In REPL, the underscore variable is used to get the last result.

1. C:\Nodejs_WorkSpace>node
2. > var x = 10
3. undefined
4. > var y = 20
5. undefined
6. > x + y
7. 30
8. > var sum = _

```
9.  undefined
10. > console.log(sum)
11. 30
12. undefined
13. >
```

36) Does Node.js supports cryptography?

Yes, Node.js Crypto module supports cryptography. It provides cryptographic functionality that includes a set of wrappers for open SSL's hash HMAC, cipher, decipher, sign and verify functions. For example:

```
const crypto = require('crypto');
const secret = 'abcdefg';
const hash = crypto.createHmac('sha256', secret)
                   .update('Welcome to JavaTpoint')
                   .digest('hex');
console.log(hash);
```

37) What is npm? What is the main functionality of npm?

npm stands for Node Package Manager. Following are the two main functionalities of npm:

- Online repositories for node.js packages/modules which are searchable on search.nodejs.org
- Command line utility to install packages, do version management and dependency management of Node.js packages.

38) What tools can be used to assure a consistent style in Node.js?

Following is a list of tools that can be used in developing code in teams, to enforce a given style guide and to catch common errors using static analysis.

- JSLint
- JSHint
- ESLint
- JSCS

39) What is the difference between operational and programmer errors?

Operational errors are not bugs, but create problems with the system like request timeout or hardware failure. On the other hand, programmer errors are actual bugs.

40) What is the difference between the global installation of dependencies and local installation of dependencies?

- Global installation of dependencies is stored in /npm directory. While local installation of dependencies stores in the local mode. Here local mode refers to the package installation in node_modules directory lying in the folder where Node application is present.
- Globally deployed packages cannot be imported using require() in Node application directly. On the other hand, locally deployed packages are accessible via require().
- To install a Node project globally -g flag is used.

```
C:\Nodejs_WorkSpace>npm install express ?g
```

- To install a Node project locally, the syntax is:

```
C:\Nodejs_WorkSpace>npm install express
```

41) What is the use of a buffer class in Node.js?

The Node.js provides Buffer class to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap. It is a global class and can be accessed in an application without importing a buffer module. Buffer class is used because pure JavaScript is not compatible with binary data. So, when dealing with TCP streams or the file system, it's necessary to handle octet streams.

42) What is the role of assert in Node.js?

The Node.js Assert is a way to write tests. It provides no feedback when running your test unless one fails. The assert module provides a simple set of assertion tests that can be used to test invariants. The module is intended for internal use by Node.js, but can be used in application code via require ('assert'). For example:

```
var assert = require('assert');
function add (a, b) {
  return a + b;
}
var expected = add(1,2);
assert( expected === 3, 'one plus two is three');
```

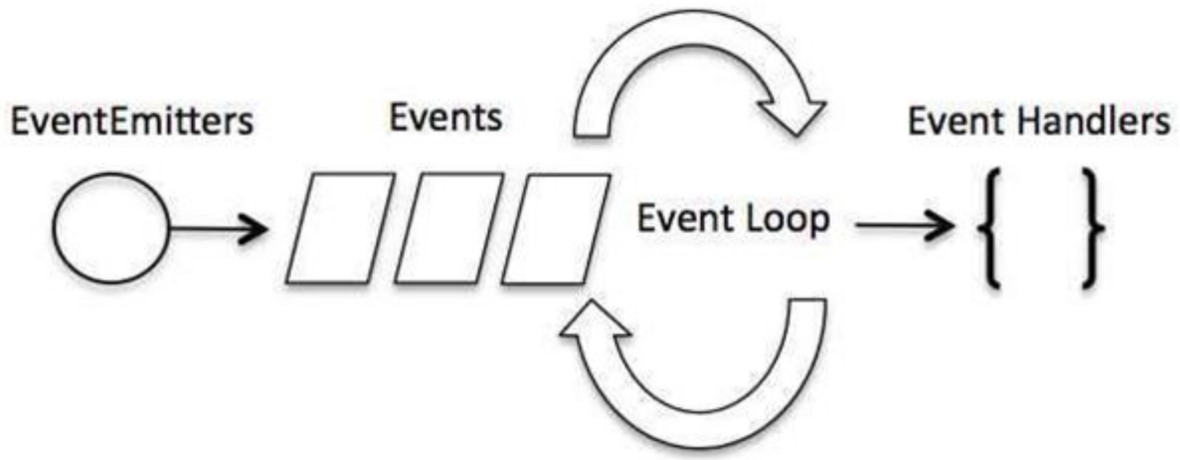
43) What are the streams in Node.js?

The Streams are the objects that facilitate you to read data from a source and write data to a destination. There are four types of streams in Node.js:

- **Readable:** This stream is used for reading operations.
- **Writable:** This stream is used for write operations.
- **Duplex:** This stream can be used for both reading and write operations.
- **Transform:** It is a type of duplex stream where the output computes according to input.

44) What is event-driven programming in Node.js?

In Node.js, event-driven programming means as soon as Node starts its server, it initiates its variables, declares functions and then waits for an event to occur. It is one of the reasons why Node.js is pretty fast compared to other similar technologies.



45) What is the difference between events and callbacks in Node.js?

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

46) What is the Punycode in Node.js?

The Punycode is an encoding syntax which is used to convert Unicode (UTF-8) string of characters to ASCII string of characters. It is bundled with Node.js v0.6.2 and later versions. If you want to use it with other Node.js versions, then use npm to install Punycode module first. You have to used require ('Punycode') to access it.

Syntax:

```
punycode = require('punycode');
```

47) What does Node.js TTY module contains?

The Node.js TTY module contains tty.ReadStream and tty.WriteStream classes. In most cases, there is no need to use this module directly. You have to used require ('tty') to access this module.

Syntax:

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

49) What are the main differences between operational and programmer errors?

The most crucial difference between operational and programmer errors is that the operational errors are not bugs but problems with the system such as to request timeout or hardware failure. On the other hand, the programmer errors are actual bugs in the application.

50) What do you understand by an EventEmitter in Node.js?

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

Example:

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

51) What is the difference between readFile and createReadStream in Node.js?

In Node.js, there are two ways to read and execute files: readFile and CreateStream.

- The readFile() process is a fully buffered process that returns the response only when the complete file is pushed into the buffer and is read. This process is called a memory-intensive process, and in the case of large files, the processing can be very slow.
- On the other hand, the createReadStream() is a partially buffered process that treats the entire process as an event series. The entire file is split into chunks and then processed and sent back as a response one by one. After completing this

step, they are finally removed from the buffer. Unlike the readFile process, the createReadStream process is effective for the processing of large files.

52) What is the concept of Punycode in Node.js?

In Node.js, the concept of Punycode is used for converting one type of string into another type. Punycode is an encoding syntax used for converting Unicode (UTF-8) string of characters into a basic ASCII string of characters. Now, the hostnames can only understand the ASCII characters so, after the Node.js version 0.6.2 onwards, it was bundled up with the default Node package.

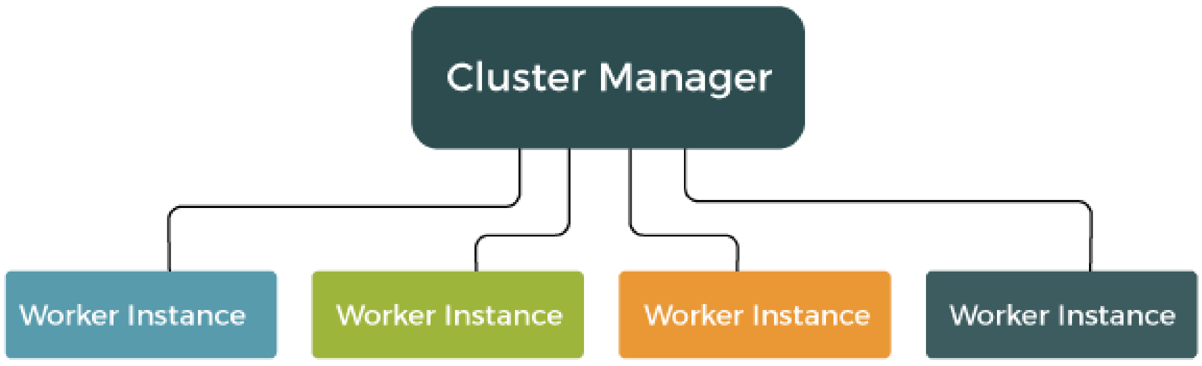
To use it with any previous versions, you have to use the following code:

Syntax:

```
1. punycode = require('punycode');
```

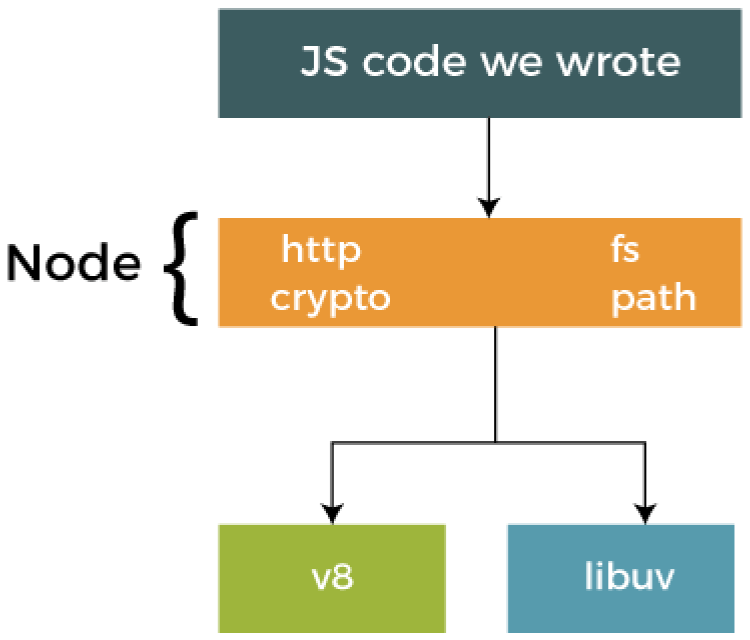
53) How can you enhance the Node.js performance through clustering?

Just because the Node.js applications run on a single processor, they don't take advantage of a multiple-core system by default. Clustering is used to overcome this issue. The cluster mode is used to start up multiple node.js processes, thereby having multiple instances of the event loop. When we start using clusters in a Node.js app, it creates multiple node.js processes. But there is also a parent process called the cluster manager, which is responsible for monitoring the health of the individual instances of the application.



54) What is a thread pool in Node.js? Which library handles it?

In Node.js, the libuv library is used to handle the Thread pool. The libuv library is a multi-platform C library that supports asynchronous I/O-based operations such as file systems, networking, and concurrency.



1. Differentiate between JavaScript and Node.js.

JavaScript vs Node.js

Features	**JavaScript**	**Node.js**
Type	Programming Language	Interpreter and environment for JavaScript
Utility	Used for any client-side activity for a web application	Used for accessing or performing any non-blocking operation of any operating system
Running Engine	Spider monkey (FireFox), JavaScript Core (Safari), V8 (Google Chrome), etc.	V8 (Google Chrome)

2. What Is Node.js?

Node.js is an extremely powerful framework developed on **Chrome's V8 JavaScript engine** that compiles the JavaScript directly into the native machine code. It is a lightweight framework used for creating server-side web applications and extends JavaScript API to offer usual server-side functionalities. It is generally used for large-scale application development, especially for video streaming sites, single page application, and other web applications.

3. List down the major benefits of using Node.js?

Features	Description
Fast	Node.js is built on Google Chrome's V8 JavaScript Engine which makes its library very fast in code execution
Asynchronous	Node.js based server never waits for an API to return data thus making it asynchronous
Scalable	It is highly scalable because of its event mechanism which helps the server to respond in a non-blocking way
Open Source	Node.js has an extensive open source community which has contributed in producing some excellent modules to add additional capabilities to Node.js applications
No Buffering	Node.js applications simply output the data in chunks and never buffer any data

4. What is the difference between Angular and Node.js?

Angular	Node.js
1. It is an **open source** web application development framework	1. It is a cross-platform **run-time environment** for applications
2. It is written in TypeScript	2. It is written in **C, C++ and JavaScript** languages
3. Used for building **single-page** client-side web applications	3. Used for building f**ast and scalable** server-side networking applications
4. Angular itself is a **web application framework**	4. Node.js has many **different frameworks** like Sails.js, Partial.js, and Express.js, etc.
5. Ideal for creating **highly active and interactive** web apps	5. Ideal for developing **small size** projects
6. Helpful in **splitting an app into MVC components**	6. Helpful in **generating database queries**

7. Suitable for developing **real-time applications**	7. Suitable in situations where something **faster and more scalable** is required

5. Why Node.js is single threaded?

Node.js uses a single threaded model in order to support async processing. With async processing, an application can perform better and is more scalable under web loads. Thus, Node.js makes use of a single-threaded model approach rather than typical thread-based implementation.

6. How do Node.js works?

Node.js is a virtual machine that uses JavaScript as its scripting language and runs on a v8 environment. It works on a single-threaded event loop and a non-blocking I/O which provides high rate as it can handle a higher number of concurrent requests. Also, by making use of the 'HTTP' module, Node.js can run on any stand-alone web server.

7. Where Node.js can be used?

Node.js can be used to develop:

- Real-Time Web Applications
- Network Applications
- Distributed Systems
- General Purpose Applications

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

There are two types of API functions in Node.js:

- Asynchronous, non-blocking functions
- Synchronous, blocking functions

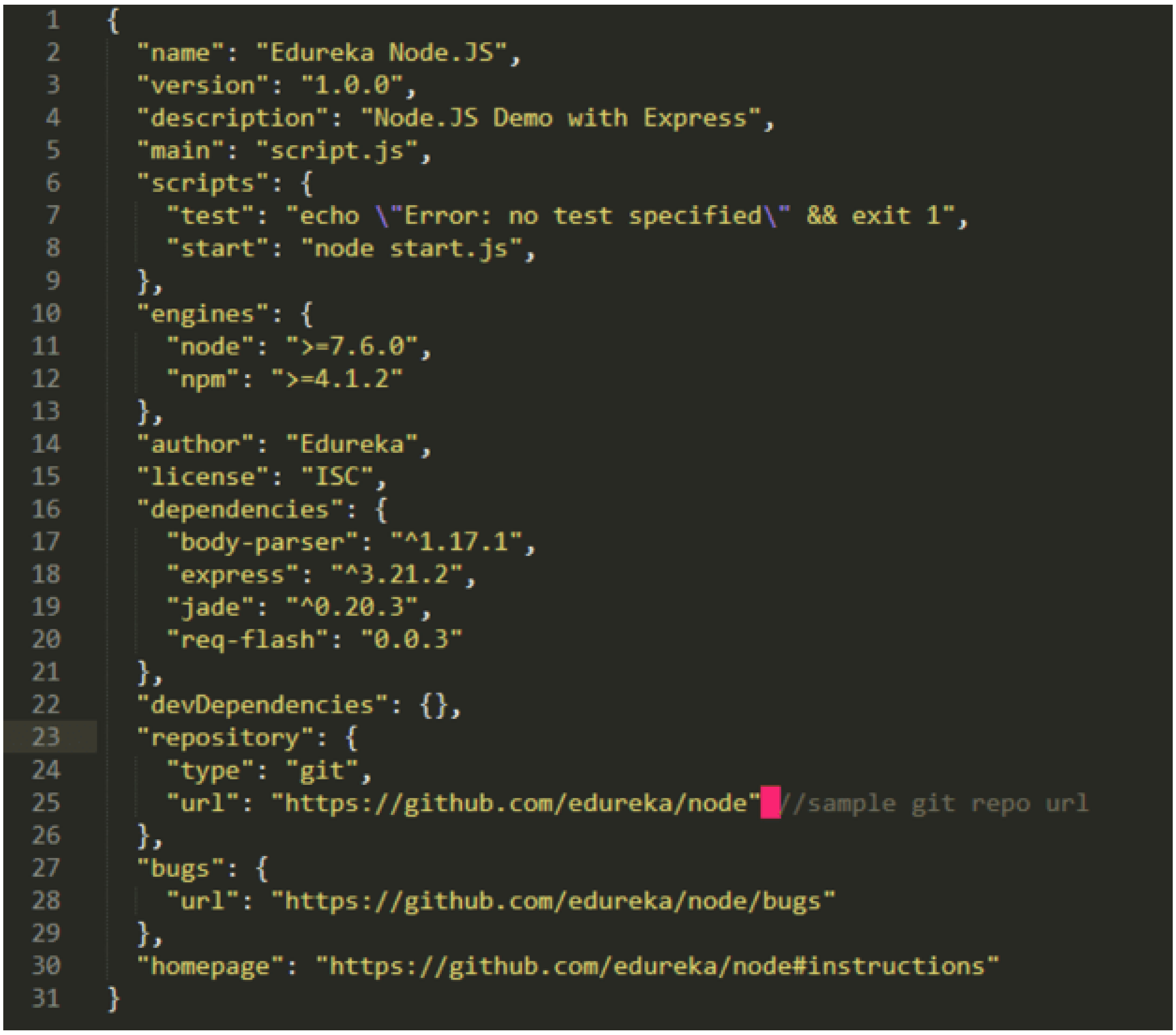
9. **What is the difference between Asynchronous and Non-blocking?**

Asynchronous	Non-blocking
Asynchronous means not synchronous. Using these we can make asynchronous HTTP requests that do not wait for the server to respond. These functions continue to respond to the request for which it has already received the server response.	Non-blocking functions are used in regards with I/O operations. They immediately respond with whatever data is available and keeps on running as per the requests. In case, any answer couldn't be retrieved then the API returns immediately with an error.

In case you are facing any challenges with these Node.js Interview Questions, please mention your problems in the section comment section below.

10. What is package.json?

The **package.json file** in Node.js is the heart of the entire application. It is basically the manifest file that contains the metadata of the project where we define the properties of a package.



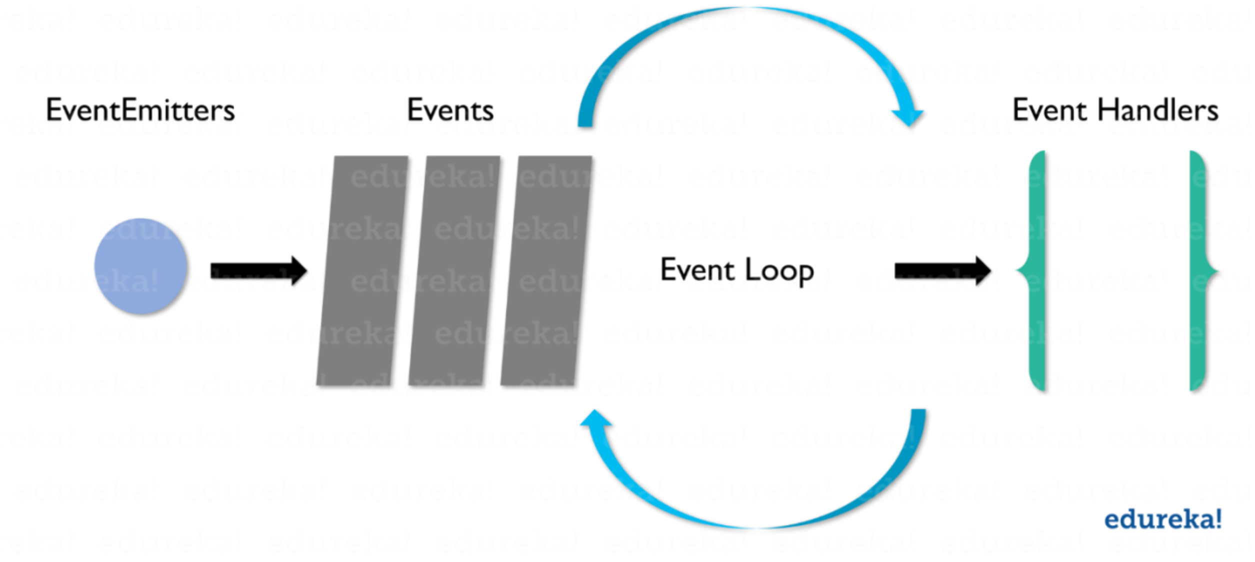
```
{
  "name": "Edureka Node.JS",
  "version": "1.0.0",
  "description": "Node.JS Demo with Express",
  "main": "script.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node start.js",
  },
  "engines": {
    "node": ">=7.6.0",
    "npm": ">=4.1.2"
  },
  "author": "Edureka",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.17.1",
    "express": "^3.21.2",
    "jade": "^0.20.3",
    "req-flash": "0.0.3"
  },
  "devDependencies": {},
  "repository": {
    "type": "git",
    "url": "https://github.com/edureka/node" //sample git repo url
  },
  "bugs": {
    "url": "https://github.com/edureka/node/bugs"
  },
  "homepage": "https://github.com/edureka/node#instructions"
}
```

11. What do you understand by Event-driven programming?

Event-driven programming is an approach that heavily makes use of events for triggering various functions. An event can be anything like a mouse click, key press, etc. When an event occurs, a call back function is executed that is already registered with the element. This approach mainly follows the publish-subscribe pattern. Because of event-driven programming, Node.js is faster when compared to other technologies.

12. What is an *Event loop* in Node.js and how does it work?

An event loop in Node.js handles all the asynchronous callbacks in an application. It is one of the most important aspects of Node.js and the reason behind Node.js have non-blocking I/O. Since Node.js is an event-driven language, you can easily attach a listener to an event and then when the event occurs the callback will be executed by the specific listener. Whenever functions like setTimeout, http.get, and fs.readFile are called, Node.js executed the event loop and then proceeds with the further code without waiting for the output. Once the entire operation is finished, Node.js receives the output and then executes the callback function. This is why all the callback functions are placed in a queue in a loop. Once the response is received, they are executed one by one.



13. Explain REPL in the context of Node.js.

REPL in Node.js stands for **R**ead, **E**val, **P**rint, and **L**oop. It represents a computer environment such as a window console or Unix/Linux shell where any command can be entered and then the system can respond with an output. Node.js comes bundled with a REPL environment by default. REPL can perform the below-listed tasks:

- **Read:** Reads the user's input, parses it into JavaScript data-structure and then stores it in the memory.
- **Eval:** Receives and evaluates the data structure.
- **Print:** Prints the final result.
- **Loop:** Loops the provided command until *CTRL+C* is pressed twice.

14. List down the tasks which should be done asynchronously using the event loop?

Below is the list of the tasks which must be done asynchronously using the event loop:

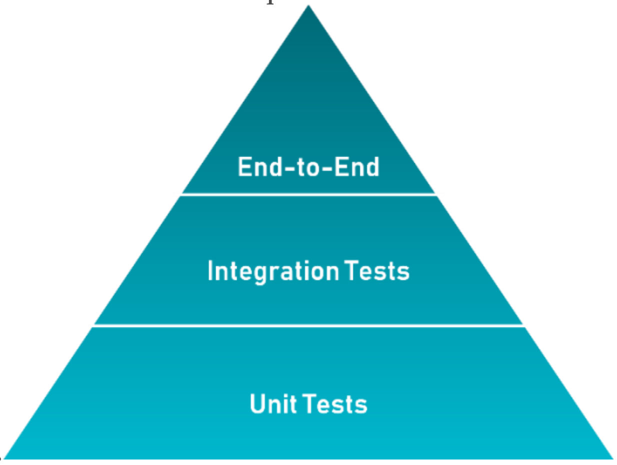
- I/O operations
- Heavy computation
- Anything requiring blocking

15. List down the steps using which "Control Flow" controls the function calls in Node.js?

1. Control the order of execution
2. Collect data
3. Limit concurrency
4. Call the next step in the program

16. What do you understand by a test pyramid?

A test pyramid basically is a diagram that describes the ratio of how many unit tests, integration tests, and end-to-end test are required to be written for the successful development of the project.



17. What is an error-first callback in Node.js?

Error-first callbacks in Node.js are used to pass errors and data. The very first parameter you need to pass to these functions has to be an error object while the other parameters represent the associated data. Thus you can pass the error object for checking if anything is wrong and handle it. In case there is no issue, you can just go ahead and with the subsequent arguments.

```
var myPost = new Post({title: 'edureka'});
myPost.save(function(err,myInstance){
if(err){
//handle error and return
}
//go ahead with `myInstance`
});
```

18. Explain the purpose of module.exports?

A module in Node.js is used to encapsulate all the related codes into a single unit of code which can be interpreted by shifting all related functions into a single file. For example, suppose you have a file called greet.js that contains the two functions as shown below:

```
module.exports = {
greetInHindi: function(){
return "NAMASTE";
},
greetInKorean: function(){
return "ANNYEONGHASEYO";
}};
```

As you can see module.exports provide two functions which can be imported in another file using below code:

```
var eduGreets = require ("./greet.js");
eduGreets.greetInHindi() //NAMASTE
eduGreets.greetInKorean() //ANNYEONGHASEYO
```

19. What do you understand by Reactor Pattern in Node.js?

Reactor Pattern in Node.js is basically a concept of non-blocking I/O operations. This pattern provides a handler that is associated with each I/O operation and as soon as an I/O request is generated, it is then submitted to a *demultiplexer*. This demultiplexer is a notification interface which is capable of handling concurrency in non-blocking I/O mode. It also helps in collecting each and every request in the form of an event and then place each event in a queue. Thus resulting in the generation of the Event Queue. Simultaneously, we have our event loop which iterates the events present in the Event Queue.

20. What's the difference between 'front-end' and 'back-end' development?

Front-end Development	Back-end Development
1. Uses mark up and web languages like HTML, CSS, JavaScript	1. Uses programming and scripting languages like Python, Ruby, Perl, etc.
2. Based on asynchronous requests and AJAX	2. Based on Server Architecture
3. Better Accessibility	3. Enhanced Security
4. Used for SEO	4. Used for Backup

21. What are LTS releases of Node.js?

LTS stands **L**ong **T**erm **S**upport version of Node.js that receives all the critical bug fixes along with security updates and performance improvements. These versions are supported for at least 18 months and mainly focus on stability and security. The modifications done to the LTS versions are restricted to the bug fixes, security upgrade, npm, and documentation updates, performance improvement, etc.

22. List down the major security implementations within Node.js?

Major security implementations in Node.js are:

1. Authentications
2. Error Handling

23. What do you understand by callback hell?

Callback Hell is also known as the Pyramid of Doom. It is a pattern caused by intensively nested callbacks which are unreadable and unwieldy. It typically contains multiple nested callback functions which in turn make the code hard to read and debug. It is caused by improper implementation of the asynchronous logic.

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

In case you are facing any challenges with these Node.js Interview Questions, please mention your problems in the section comment section below.

24. Explain libuv.

Libuv is a multi-platform support library of Node.js which majorly is used for asynchronous I/O. It was primarily developed for Node.js, with time it is popularly practiced with other systems like as Luvit, pyuv, Julia, etc. Libuv is basically an abstraction around libev/ IOCP depending on the platform, providing users an API based on libev. A few of the important features of libuv are:

- Full-featured event loop backed
- File system events
- Asynchronous file & file system operations
- Asynchronous TCP & UDP sockets
- Child processes

25. Explain the concept of middleware in Node.js?

In general, middleware is a function receives the Request and Response objects. In other words, in an application's request-response cycle these functions have access to various request & response objects along with the next function of the cycle. The next function of middleware is represented with the help of a variable, usually named next. Most commonly performed tasks by the middleware functions are:

- Execute any type of code
- Update or modify the request and the response objects
- Finish the request-response cycle
- Invoke the next middleware in the stack

26. Explain the concept of URL module.

The **URL module** of Node.js provides various utilities for **URL** resolution and parsing. It is a built-in module that helps in splitting up the web address into a readable format:

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

For example:

```
var url = require('url');
var adrs = '<a href="http://localhost:8082/default.htm?year=2019&amp;month=april">http://localh
var q = url.parse(adr, true);
console.log(q.host); //returns 'localhost:8082'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2019 and month=april'
var qdata = q.query; //returns an object: { year: 2019, month: 'april' }
console.log(qdata.month); //returns 'april'
```

27. What do you understand by ESLint?

ESLint is an open source project initially developed by Nicholas C. Zakas in 2013 which aims to provide a linting utility for JavaScript through a plug. Linters in Node.js are good tools for searching certain bug classes, especially those which are related to the variable scope.

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

Google uses V8 as it is a Chrome runtime engine that converts JavaScript code into native machine code. This, in turn, speeds up the application execution and response process and give you a fast running application.

29. Explain the working of the control flow function.

In Node.js, the control flow function is basically the code that is executed between the asynchronous function calls. Below are the steps that must be followed for executing it:

1. Firstly, the order of execution must be controlled.
2. Then, the required data need to be collected.
3. Next, the concurrency must be limited.
4. Once done, the next step of the program has to be invoked.

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

Below are the two arguments that async.queue takes as input:

1. Task Function
2. Concurrency Value

31. Differentiate between spawn() and fork() methods in Node.js?

In Node.js, the spawn() is used to launch a new process with the provided set of commands. This method doesn't create a new V8 instance and just one copy of the node module is active on the processor. When your child process returns a large amount of data to the Node you can invoke this method.

Syntax:

```
child_process.spawn(command[, args][, options])
```

Whereas, the fork() in Node.js is a special instance of spawn() that executes a new instance of the V8 engine. This method simply means that multiple workers are running on a single Node code base for various task.

Syntax:

```
child_process.fork(modulePath[, args][, options])
```

In case you are facing any challenges with these Node.js Interview Questions, please mention your problems in the section comment section below.

32. What do you understand by global objects in Node.js?

In Node.js, Globals are the objects which are global in nature and are available in all the modules of the application. You can use these objects directly in your application, rather than having to include them explicitly. The global objects can be modules, functions, strings, object, etc. Moreover, some of these objects can be in the module scope instead of global scope.

33. Explain the concept of stub in Node.js.

In Node.js, stubs are basically the programs or functions that are used for stimulating the module or component behavior. During any test cases, stubs provide the canned answers of the functions.

34. How assert works in Node.js?

In Node.js, assert is used to write tests. It only provides feedback only when any of the running test cases fails. This module gives you a set of assertion tests which are then used for testing invariants. It is basically used internally by Node.js but using require('assert') code, it can be used in other applications as well.

```
var assert = require('assert');
function mul(a, b) {
return a * b;
}
var result = mul(1,2);
assert( result === 2, 'one multiplied by two is two');
```

35. Define the concept of the test pyramid. Explain the process to implement them in terms of HTTP APIs.

The test **pyramid** is basically a concept that is developed by Mike Cohn. According to this, you should have a higher number of low-level unit **tests** as compared to high-level end-to-end **tests that** running through a GUI.

In terms of HTTP APIs it may be defined as:

- A higher number of low-level unit tests for each model
- Lesser integration tests to test model interactions
- Lesser acceptance tests for testing actual HTTP endpoints

36. Explain the purpose of ExpressJS package?

Express.js is a framework built on top of Node.js that facilitates the management of the flow of data between server and routes in the server-side applications. It is a lightweight and flexible framework that provides a wide range of features required for the web as well as mobile application development. Express.js is developed on the middleware module of Node.js called ***connect***. The connect module further makes use of **http** module to communicate with Node.js. Thus, if you are working with any of the connect based middleware modules, then you can easily integrate with Express.js.

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

In Node.js, process.nextTick() and setImmediate(), both are functions of the Timers module which help in executing the code after a predefined period of time. But these functions differ in their execution. The process.nextTick function waits for the execution of action till the next pass around in the event loop or once the event loop is completed only then it will invoke the callback function. On the other hand, setImmediate() is used to execute a callback method on the next cycle of the event loop which eventually returns it to the event loop in order to execute the I/O operations.

38. Explain the usage of a buffer class in Node.js?

Buffer class in Node.js is used for storing the raw data in a similar manner of an array of integers. But it corresponds to a raw memory allocation that is located outside the V8 heap. It is a global class that is easily accessible can be accessed in an application without importing a buffer module. Buffer class is used because pure JavaScript is not compatible with binary data. So, when dealing with TCP streams or the file system, it's necessary to handle octet streams.

39. How does Node.js handle the child threads?

In general, Node.js is a single threaded process and doesn't expose the child threads or thread management methods. But you can still make use of the child threads using spawn() for some specific asynchronous I/O tasks which execute in the background and don't usually execute any JS code or hinder with the main event loop in the application. If you still want to use the threading concept in your application you have to include a module called ChildProcess explicitly.

40. Explain stream in Node.js along with its various types.

Streams in Node.js are the collection of data similar to arrays and strings. They are objects using which you can read data from a source or write data to a destination in a continuous manner. It might not be available at once and need not to have fit in the memory. These streams are especially useful for reading and processing a large set of data. In Node.js, there are four fundamental types of streams:

1. *Readable:* Used for reading large chunks of data from the source.
2. *Writeable:* Use for writing large chunks of data to the destination.
3. *Duplex:* Used for both the functions; read and write.
4. *Transform:* It is a duplex stream that is used for modifying the data.

41. What is the use of NODE_ENV?

If the project is in the production stage, Node.js promotes the convention of making use of NODE_ENV variable to flag it. This helps in taking better judgment during the development of the projects. Also, when you set your NODE_ENV to production, your application tends to perform 3 times faster.

42. Differentiate between readFile vs createReadStream in Node.js?

Node.js provides two ways to read and execute files which are using readFile and CreateStream. readFile() is a fully buffered process which returns the response only when the complete file is pushed into the buffer and is read. It is a memory intensive process and in case of large files, the processing can be very slow. Whereas createReadStream is a partially buffered which treats the entire process as an event series. The entire file is split into chunks which are then processed and sent back as a response one by one. Once done, they are finally removed from the buffer. Unlike readFile, createReadStream is really effective for the processing of the large files.

43. List down the various timing features of Node.js.

Node.js provides a Timers module which contains various functions for executing the code after a specified period of time. Below I have listed down the various functions provided by this module:

- **setTimeout/clearTimeout** – Used to schedule code execution after a designated amount of milliseconds
- **setInterval/clearInterval** – Used to execute a block of code multiple times
- **setImmediate/clearImmediate** – Used to execute code at the end of the current event loop cycle
- **process.nextTick** – Used to schedule a callback function that needs to be invoked in the next iteration of the Event Loop

44. Explain the concept of Punycode in Node.js?

In Node.js, Punycode is an encoding syntax that is used for converting Unicode (UTF-8) string of characters into a basic ASCII string of characters. It is important as the hostnames can only understand the ASCII characters. Thus, Node.js version 0.6.2 onwards, it was bundled up with the default Node package. If you want to use it with any previous versions, you can easily do that by using the following code:

Syntax:

```
punycode = require('punycode');
```

45. Differentiate between Node.js vs Ajax?

The most basic difference between Node.js and Ajax that, Node.js is a server-side JavaScript whereas Ajax is a client-side technology. In simpler terms, Ajax is mostly used for updating or modifying the webpage contents without having to refresh it. On the other hand, Node.js is required to develop the server software that are typically executed by the servers instead of the web browsers.

46. Does Node.js provide any Debugger?

Node.js do provide a simple TCP based protocol and debugging client that comes built-in. In order to debug your JavaScript file, you can use the below debug argument followed by **js** file name that you want to debug.

Syntax:

```
node debug [script.js | -e "script" | &lt;host&gt; : &lt;port&gt; ]
```

In case you are facing any challenges with these Node.js Interview Questions, please mention your problems in the section comment section below.

47. Describe the exit codes of Node.js.

In Node.js, exit codes are a set of specific codes which are used for finishing a specific process. These processes can include the global object as well. Below are some of the exit codes used in Node.js:

- Uncaught fatal exception
- Unused
- Fatal Error
- Internal Exception handler Run-time failure
- Internal JavaScript Evaluation Failure

48. What do you understand by an Event Emitter in Node.js?

EventEmitter is a Node.js class that includes all the objects that are capable of emitting events. These objects contain an eventEmitter.on() function through which more than one function can be attached to the named events that are emitted by the object. Whenever an EventEmitter object throws an event, all the attached functions to that specific event are invoked synchronously. Below code shows how to us the EventEmitter in your application:

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

49. Is cryptography supported in Node.js?

Yes, Node.js does support cryptography through a module called Crypto. This module provides various cryptographic functionalities like cipher, decipher, sign and verify functions, a set of wrappers for open SSL's hash HMAC etc. For example:

Syntax:

```
const crypto = require'crypto');
const secret = 'akerude';
const hash = crypto.createHmac('swaEdu', secret).update('Welcome to Edureka').digest('hex');
console.log(hash);
```

50. Explain the reason as to why Express 'app' and 'server' must be kept separate.

Express 'app' and 'server' must be kept separate as by doing this, you will be separating the API declaration from the network related configuration which benefits in the below listed ways:

- It allows testing the API in-process without having to perform the network calls
- Faster testing execution
- Getting wider coverage metrics of the code
- Allows deploying the same API under flexible and different network conditions
- Better separation of concerns and cleaner code

API declaration should reside in app.js:

```
var app = express();
app.use(bodyParser.json());
app.use("/api/events", events.API);
app.use("/api/forms", forms);
```

Server network declaration should reside in /bin/www:

```
var app = require('../app');
var http = require('http');
//Get port from environment and store in Express
```

```
var port = normalizePort(process.env.PORT || '8000');
app.set('port', port);
//Create HTTP server.
var server = http.createServer(app);
```

What is Node.js?

Node.js is a web application framework built on Google Chrome's JavaScript Engine(V8 Engine).

Node.js comes with runtime environment on which a Javascript based script can be interpreted and executed (It is analogus to JVM to JAVA byte code). This runtime allows to execute a JavaScript code on any machine outside a browser. Because of this runtime of Node.js, JavaScript is now can be executed on server as well.

Node.js also provides a rich library of various javascript modules which eases the developement of web application using Node.js to great extents.

Node.js = Runtime Environment + JavaScript Library

What do you mean by Asynchronous API?

All APIs of Node.js library are asynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.

What are the benefits of using Node.js?

Following are main benefits of using Node.js

Asynchronous and Event DrivenAll APIs of Node.js library are asynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get 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 server to respond in a non-bloking ways and makes server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and same program can services much larger number of requests than traditional server like Apache HTTP Server.

No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.

Is it free to use Node.js?

Yes! Node.js is released under the MIT license and is free to use.

Is Node a single threaded application?

Yes! Node uses a single threaded model with event looping.

What is REPL in context of Node?

REPL stands for Read Eval Print Loop and it represents a computer environment like a window console or unix/linux shell where a command is entered and system responds with an output. Node.js or Node comes bundled with a REPL environment. It performs the following desired tasks.

Read − Reads user's input, parse the input into JavaScript data-structure and stores in memory.

Eval − Takes and evaluates the data structure

Print − Prints the result

Loop − Loops the above command until user press ctrl-c twice.

Can we evaluate simple expression using Node REPL

Yes.

What is the difference of using var and not using var in REPL while dealing with variables?

Use variables to store values and print later. if var keyword is not used then value is stored in the variable and printed. Whereas if var keyword is used then value is stored but not printed. You can use both variables later.

What is the use of Underscore variable in REPL?

Use _ to get the last result.

```
C:\Nodejs_WorkSpace>node
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

What is npm?

npm stands for Node Package Manager. npm provides following two main functionalities:

Online repositories for node.js packages/modules which are searchable on search.nodejs.org

Command line utility to install packages, do version management and dependency management of Node.js packages.

What is global installation of dependencies?

Globally installed packages/dependencies are stored in <user-directory>/npm directory. Such dependencies can be used in CLI (Command Line Interface) function of any node.js but can not be imported using require() in Node application directly. To install a Node project globally use -g flag.

```
C:\Nodejs_WorkSpace>npm install express -g
```

What is local installation of dependencies?

By default, npm installs any dependency in the local mode. Here local mode refers to the package installation in node_modules directory lying in the folder where Node application is present. Locally deployed packages are accessible via require(). To install a Node project locally following is the syntax.

```
C:\Nodejs_WorkSpace>npm install express
```

How to check the already installed dependencies which are globally installed using npm?

Use the following command −

```
C:\Nodejs_WorkSpace>npm ls -g
```

What is Package.json?

package.json is present in the root directory of any Node application/module and is used to define the properties of a package.

Name some of the attributes of package.json?

Following are the attributes of Package.json

name − name of the package

version − version of the package

description − description of the package

homepage − homepage of the package

author − author of the package

contributors − name of the contributors to the package

dependencies − list of dependencies. npm automatically installs all the dependencies mentioned here in the node_module folder of the package.

repository − repository type and url of the package

main − entry point of the package

keywords − keywords

How to uninstall a dependency using npm?

Use following command to uninstall a module.

```
C:\Nodejs_WorkSpace>npm uninstall dependency-name
```

How to update a dependency using npm?

Update package.json and change the version of the dependency which to be updated and run the following command.

```
C:\Nodejs_WorkSpace>npm update
```

What is Callback?

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All APIs of Node are written is such a way that they supports callbacks. For example, a function to read a file may start reading file and return the control to execution environment immidiately so that next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process high number of request without waiting for any function to return result.

What is a blocking code?

If application has to wait for some I/O operation in order to complete its execution any further then the code responsible for waiting is known as blocking code.

How Node prevents blocking code?

By providing callback function. Callback function gets called whenever corresponding event triggered.

What is Event Loop?

Node js is a single threaded application but it support concurrency via concept of event and callbacks. As every API of Node js are asynchronous and being a single thread, it uses async function calls to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever any task get completed, it fires the corresponding event which signals the event listener function to get executed.

What is Event Emmitter?

EventEmitter class lies in events module. It is accessibly via following syntax −

```
//import events module
var events = require('events');

//create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

When an EventEmitter instance faces any error, it emits an 'error' event. When new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener' event is fired.

EventEmitter provides multiple properties like on and emit. on property is used to bind a function with the event and emit is used to fire an event.

What is purpose of Buffer class in Node?

Buffer class is a global class and can be accessed in application without importing buffer module. A Buffer is a kind of an array of integers and corresponds to a raw memory allocation outside the V8 heap. A Buffer cannot be resized.

What is Piping in Node?

Piping is a mechanism to connect output of one stream to another stream. It is normally used to get data from one stream and to pass output of that stream to another stream. There is no limit on piping operations. Consider the above example, where we've read test.txt using readerStream and write test1.txt using writerStream. Now we'll use the piping to simplify our operation or reading from one file and writing to another file.

Which module is used for file based operations?

fs module is used for file based operations.

var fs = require("fs")

Which module is used for buffer based operations?

buffer module is used for buffer based operations.

var buffer = require("buffer")

Which module is used for web based operations?

http module is used for web based operations.

var http = require("http")

fs module provides both synchronous as well as asynchronous methods.

true.

What is difference between synchronous and asynchronous method of fs module?

Every method in fs module have synchronous as well as asynchronous form. Asynchronous methods takes a last parameter as completion function callback and first parameter of the callback function is error. It is preferred to use asynchronous method instead of synchronous method as former never block the program execution where the latter one does.

Name some of the flags used in read/write operation on files.

flags for read/write operations are following:

r − Open file for reading. An exception occurs if the file does not exist.

r+ − Open file for reading and writing. An exception occurs if the file does not exist.

rs − Open file for reading in synchronous mode. Instructs the operating system to bypass the local file system cache. This is primarily useful for opening files on NFS mounts as it allows you to skip the potentially stale local cache. It has a very real impact on I/O performance so don't use this flag unless you need it. Note that this doesn't turn fs.open() into a synchronous blocking call. If that's what you want then you should be using fs.openSync()

rs+ − Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' about using this with caution.

w − Open file for writing. The file is created (if it does not exist) or truncated (if it exists).

wx − Like 'w' but fails if path exists.

w+ − Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).

wx+ − Like 'w+' but fails if path exists.

a − Open file for appending. The file is created if it does not exist.

ax − Like 'a' but fails if path exists.

a+ − Open file for reading and appending. The file is created if it does not exist.

ax+' − Like 'a+' but fails if path exists.

What are streams?

Streams are objects that let you read data from a source or write data to a destination in continuous fashion.

How many types of streams are present in Node.

In Node.js, there are four types of streams.

Readable − Stream which is used for read operation.

Writable − Stream which is used for write operation.

Duplex − Stream which can be used for both read and write operation.

Transform − A type of duplex stream where the output is computed based on input.

Name some of the events fired by streams.

Each type of Stream is an EventEmitter instance and throws several events at different instance of times. For example, some of the commonly used events are:

data − This event is fired when there is data is available to read.

end − This event is fired when there is no more data to read.

error − This event is fired when there is any error receiving or writing data.

finish − This event is fired when all data has been flushed to underlying system

What is Chaining in Node?

Chanining is a mechanism to connect output of one stream to another stream and create a chain of multiple stream operations. It is normally used with piping operations.

How will you open a file using Node?

Following is the syntax of the method to open a file in asynchronous mode:

fs.open(path, flags[, mode], callback)

Parameters

Here is the description of the parameters used:

path − This is string having file name including path.

flags − Flag tells the behavior of the file to be opened. All possible values have been mentioned below.

mode − This sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.

callback − This is the callback function which gets two arguments (err, fd).

How will you read a file using Node?

Following is the syntax of one of the methods to read from a file:

fs.read(fd, buffer, offset, length, position, callback)

This method will use file descriptor to read the file, if you want to read file using file name directly then you should use another method available.

Parameters

Here is the description of the parameters used −

fd − This is the file descriptor returned by file fs.open() method.

buffer − This is the buffer that the data will be written to.

offset − This is the offset in the buffer to start writing at.

length − This is an integer specifying the number of bytes to read.

position − This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.

callback − This is the callback function which gets the three arguments, (err, bytesRead, buffer).

How will you write a file using Node?

Following is the syntax of one of the methods to write into a file:

fs.writeFile(filename, data[, options], callback)

This method will over-write the file if file already exists. If you want to write into an existing file then you should use another method available.

Parameters

Here is the description of the parameters used:

path − This is string having file name including path.

data − This is the String or Buffer to be written into the file.

options − The third parameter is an object which will hold {encoding, mode, flag}. By default encoding is utf8, mode is octal value 0666 and flag is 'w'

callback − This is the callback function which gets a single parameter err and used to to return error in case of any writing error.

How will you close a file using Node?

Following is the syntax of one of the methods to close an opened file:

fs.close(fd, callback)

Parameters

Here is the description of the parameters used:

fd − This is the file descriptor returned by file fs.open() method.

callback − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you get information about a file using Node?

Following is the syntax of the method to get the information about a file:

fs.stat(path, callback)

Parameters

Here is the description of the parameters used:

path − This is string having file name including path.

callback − This is the callback function which gets two arguments (err, stats) where stats is an object of fs.Stats type which is printed below in the example.

How will you truncate a file using Node?

Following is the syntax of the method to truncate an opened file −

fs.ftruncate(fd, len, callback)

Parameters

Here is the description of the parameters used:

fd − This is the file descriptor returned by file fs.open() method.

len − This is the length of the file after which file will be truncated.

callback − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you delete a file using Node?

Following is the syntax of the method to delete a file −

fs.unlink(path, callback)

Parameters

Here is the description of the parameters used:

path − This is the file name including path.

callback − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you create a directory?

Following is the syntax of the method to create a directory:

fs.mkdir(path[, mode], callback)

Parameters

Here is the description of the parameters used:

path − This is the directory name including path.

mode − This is the directory permission to be set. Defaults to 0777.

callback − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you delete a directory?

Following is the syntax of the method to remove a directory:

fs.rmdir(path, callback)

Parameters

Here is the description of the parameters used:

path − This is the directory name including path.

callback − This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

How will you read a directory?

Following is the syntax of the method to read a directory:

fs.readdir(path, callback)

Parameters

Here is the description of the parameters used:

path − This is the directory name including path.

callback − This is the callback function which gets two arguments (err, files) where files is an array of the names of the files in the directory excluding '.' and '..'.

What is the purpose of __filename variable?

The __filename represents the filename of the code being executed. This is the resolved absolute path of this code file. For a main program this is not necessarily the same filename used in the command line. The value inside a module is the path to that module file.

What is the purpose of __dirname variable?

The __dirname represents the name of the directory that the currently executing script resides in.

What is the purpose of setTimeout function?

The setTimeout(cb, ms) global function is used to run callback cb after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days.

This function returns an opaque value that represents the timer which can be used to clear the timer.

What is the purpose of clearTimeout function?

The clearTimeout(t) global function is used to stop a timer that was previously created with setTimeout(). Here t is the timer returned by setTimeout() function.

What is the purpose of setInterval function?

The setInterval(cb, ms) global function is used to run callback cb repeatedly after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days.

This function returns an opaque value that represents the timer which can be used to clear the timer using the function clearInterval(t).

What is the purpose of console object?

console object is used to Used to print information on stdout and stderr.

What is the purpose of process object?

process object is used to get information on current process. Provides multiple events related to process activities.

EXPRESSJS INTERVIEW QUESTIONS

1) What is Express.js?

Express.js, or simply Express, is a free, open-source, lightweight, and fast backend web application framework for Node.js. It is released as open-source software under the MIT License.

It is designed for building single-page, multi-page, and hybrid web applications and APIs. It is called the de facto standard server framework for Node.js. It was founded and developed by **TJ Holowaychuk** in 2010 and written in JavaScript.

2) What are some distinctive features of Express?

As Express is a lightweight, minimal and flexible Node.js web application framework, it provides a robust set of features for web and mobile applications. Following is the list of some distinctive features of this framework:

- js can be used to design single-page, multi-page, and hybrid web applications and APIs.
- It allows to set up middleware to respond to HTTP/RESTful Requests.
- It defines a routing table to perform different HTTP operations (method and URL).
- It allows to dynamically rendering HTML Pages based on passing arguments to templates.
- It provides high performance because of its ultra-fast I/O. It prepares a thin layer; therefore, the performance is adequate.
- Its MVC-like structure makes it organize the web application into MVC architecture.
- It provides good database support. It supports RDBMS as well as NoSQL databases.
- It is asynchronous and single-threaded.
- Its robust API makes routing easy.

3) Is Express.js front-end or backend framework?

Express.js or Express is a JavaScript backend framework. It is mainly designed to develop complete web applications (single-page, multi-page, and hybrid web applications) and APIs. Express is the backend component of the **MEAN stack** where **M stands for MongoDB**, which handles database; **E stands for Express,** which handles backend; **A stands for AngularJS**, which is for the front-end, and **N stands for Node**.

4) Why do we use Express.js?

Express.js is an automatically prebuilt Node.js framework that facilitates us to create server-side web applications faster and smarter. The main reason for choosing Express is its simplicity, minimalism, flexibility, and scalability characteristics.

5) What is the difference between Express.js and Node.js?

Node.js is an open-source, cross-platform run-time environment used for executing JavaScript code outside of a browser. Node.js is not a framework or a programming language; it is a platform that acts as a web server. Many big companies such as Paypal, Uber, Netflix, Wallmart, etc., are using this. On the other hand, Express is a small framework based on the functionality of Node.js.

Some key differences between Express.js and Node.js:

Feature	Express.js	Node.js
Definition	Express.js is a lightweight and fast backend web application framework for Node.js.	Node.js is an open-source and cross-platform that is used to execute JavaScript code outside of a browser.
Usage	Express.js is used to develop complete web applications such as single-page, multi-page, and hybrid web applications and APIs. It uses approaches and principles of Node.js.	Node.js is used to build server-side, input-output, event-driven apps.
Features	Express has more features than Node.js.	Node.js has fewer features as compared to Express.js.
Building Block	Express.js is built on Node.js.	Node.js is built on Google's V8 engine.
Written in	Express.js is written in JavaScript only.	Node.js is written in C, C++, and JavaScript language.
Framework/Platform	Express.js is a framework of Node.js based on its functionalities.	Node.js is a run-time platform or environment designed for server-side execution of JavaScript.
Controllers	Express.js is assigned with controllers.	Node.js is assigned with controllers.
Routing	Routing is provided in Express.js.	Routing is not provided in Node.js.

Middleware	Express.js uses middleware to arrange the functions systematically on the server-side.	Node.js doesn't use any such provision of middleware.
Coding	Express is easy to code and requires less coding time.	Node.js requires more coding time as compare to Express.j

6) How does an Express code look like?

The express.js program is saved with ".js" extension.

See the example:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Welcome to JavaTpoint!');
});
var server = app.listen(8000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Example app listening at http://%s:%s', host, port);
});
```

When you run the Node.js command prompt, the app will listen at the specified server address and give the following output.

Output:

```
Welcome to JavaTpoint!
```

7) Write a code to get post a query in Express.js.

```
var bodyParser = require('body-parser')
app.use( bodyParser.json() );       // to support JSON-encoded
app.use(bodyParser.urlencoded({     // to support URL-encoded
  extended: true
}));
```

8) What do you understand by Scaffolding in Express.js?

Scaffolding is a technique used for creating the skeleton structure of an application. It facilitates users to easily create their public directories, routes, views, etc., or a web application skeleton. Generally, users manually create their public directory, add middleware, create separate route files, etc. Using a scaffolding tool, they can set up all these things to directly get started with building their application.

There are two ways to install Scaffolding and use it in your application.

1. Express application generator
2. Yeoman

Express application generator: This is used to create an application skeleton quickly. Use the following command to install the Express application generator.

1. npm install express-generator -g
2. express myApp

By using the above command, a project named "myApp" will be created along with following the files/folders in the project.

- **Bin:** The bin folder contains one file called www is the main configuration file of the app.
- **Public:** The public folder contains JavaScript, CSS, and images, etc.
- **Routes:** This folder contains the routing files.
- **Views:** The view folder contains the view files of the application.
- **js:** The app.js file is the main file of the application.
- **json:** The package.json file is the manifest file. It contains all metadata of the project, such as the packages used in the app (called dependencies) etc.

Now, we have to install all the dependencies mentioned in the package.json file by using the following command:

1. cd myApp
2. npm install

Yeoman: Use the following command in your terminal to install Yeoman:

1. npm install -g yeoman

Yeoman uses generators to scaffold out applications.

9) Do Other MVC frameworks also support scaffolding?

The Scaffolding technique is also supported by other MVC frameworks other than Express. The following frameworks mainly support it: Ruby on Rails, OutSystems Platform, Play framework, Django, MonoRail, Brail, Symfony, Laravel, CodeIgniter, YII, CakePHP, Phalcon PHP, Model-Glue, PRADO, Grails, Catalyst, Seam Framework, Spring Roo, ASP.NET, etc.

10) Which are the arguments available to an Express JS route handler function?

Following are the arguments that are available to an Express.js route handler-function:

- **Req:** the request object
- **Res:** the response object
- **Next (optional):** It is a function employed to pass management to one of the above route handlers.

Note: The third argument is optional and should be omitted; however, in some cases, it is helpful.

11) What is the difference between Express and Django?

Django is a standalone and lightweight web server for testing and development. On the other hand, Express.js is a Node.js framework that can set the middleware to reply to any HTTP request.

Following is a list of some key differences between Express.js and Django:

Aspects	Express.js	Django
Architecture	Express follows the MVC architecture.	Django supports the MTV (Model Template View) design. It uses managing data for interacting and validating.
Framework	Express is a free, open-source, lightweight, and fast backend web application framework for Node.js to build single-page, multi-page,	This is a Python-based framework used to develop computer apps in a specified time frame.

	and hybrid web applications and APIs.	
Efficiency	It is best for developing web applications rapidly on Node.js.	It is more efficient and delivers at a fast speed so, it is cost-effective.
Programming language	The Express framework is programmed in Node.js.	Django is programmed in Python programming language.
Complexity	Express.js is less complex than Django.	Django is more complex than Express.js
Scalability	It provides better scalability.	It is less scalable.
Flexibility	Express is a flexible, minimal API-developing Node.js tool.	It provides limited flexibility.
Full-stack development	It provides a full-stack development that reduces the cost as you don't need to hire several developers to administer a web application's backend and frontend.	It does not deliver full-stack development.
Companies using this technology	Companies such as PayPal, IBM, Fox Sports, etc., are using this technology.	Companies such as Instagram, Mozilla, Bitbucket, etc., are using this technology.

12) How can you enable debugging in Express.js app?

There are different ways to enable debugging in Express.js app in different Operating Systems

Use the following command on Linux:

1. DEBUG=express:*
2. node app.js

Use the following command on Windows:

```
set DEBUG=express:*
node app.js
```

13) How can you allow CORS in Express.js?

We can allow CORS in Express.js, by adding the following code in server.js:

```
app.all('*', function(req, res, next) {
res.set('Access-Control-Allow-Origin', '*');
res.set('Access-Control-Allow-Methods', 'GET, POST, DELETE, PUT');
res.set('Access-Control-Allow-Headers', 'X-Requested-With, Content-Type');
if ('OPTIONS' == req.method) return res.send(200);
next();
});
```

14) How can you deal with error handling in Express.js? Explain with an example.

Error handling is much easier in the Express versions over Express 4.0. Use the following steps to do the error handling:

Create an Express.js application. There is no built-in middleware like error handler in express 4.0, so you have to either install a middleware or create a custom one.

Create a Middleware:

Create a middleware as following:

```
// error handler
app.use(function(err, req, res, next) {
// set locals, only providing error in development
res.locals.message = err.message;
res.locals.error = req.app.get('env') === 'development' ? err : {};
// render the error page
res.status(err.status || 500);
res.render('error');
});
```

Install Error Handler Middleware:

Install the errorhandler as following:

```
npm install errorhandler --save
```

Create a variable:

```
var errorhandler = require('errorhandler')
```

Use the middleware as following:

```
if (process.env.NODE_ENV === 'development') {
 // only use in development
  app.use(errorhandler({log: errorNotification}))
}
function errorNotification(err, str, req) {
 var title = 'Error in ' + req.method + ' ' + req.url
 notifier.notify({
   title: title,
   message: str
 })
}
```

15) Write the code to start serving static files in Express.js.

See the Example:

```
app.use(express.static('public'))
app.use('/static', express.static(path.join(__dirname, 'public')))
```

16) What is Middleware in Express.js? What are the different types of Middleware?

Middleware is a function invoked by the Express routing layer before the final request handler.

Middleware functions are used to perform the following tasks:

- It is used to execute any code.
- It is also used to make changes to the request and the response objects.
- It is responsible for ending the request-response cycle.

- It can call the next middleware function in the stack.

Note: 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.

Type of Middleware

Following are the main types of Middleware:

- Application-level Middleware
- Router-level Middleware
- Error-handling Middleware
- Built-in Middleware
- Third-party Middleware

Application-level middleware: The application-level middleware method is used to bind to the app object using app.use() method. It applies on all routes.

```
//This middleware will execute for each route.
app.use(function (req, res, next) {
  console.log('Current Time:', Date.now())
  next()
})
```

Router-level Middleware: The router-level Middleware is used to bind to a specific instance of express.Router().Built-in Middleware: The built-in Middleware was introduced with version 4.x. It ends the dependency on Connect.

There are the following built-in middleware functions in Express.js:

- **static:** It is used to serve static assets such as HTML files, images, etc.
- **json:** It is used to parse the incoming requests with JSON payloads. It is available with Express 4.16.0+
- **urlencoded:** It is used to parse the incoming requests with URL-encoded payloads. It is available with Express 4.16.0+

Third-party Middleware: There are many third-party middleware available such as:

- Body-parser
- Cookie-parser

- Mongoose
- Sequelize
- Cors
- Express-validator

To handle HTTP POST requests in Express.js version 4 and above, we have to install a middleware module called body-parser. Body-parser extracts the entire body portion of an incoming request stream and exposes it on req.body, The Middleware was a part of Express.js earlier, but now you have to install it separately. You can install it by using the following command:

```
npm install MODULE_NAME
```

You can load it by using requires and used later:

See the Example:

```
var bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }))
```

Note: You can use multiple middleware as an array on a single route.

See the Example:

```
var middlewareArray = [middleware1, middleware2]
app.get('/home', middlewareArray, function (req, res, next) {
  //Code snippets
})
```

17) Which template engines do Express support?

Express.js supports any template engine that follows the (path, locals, callback) signature.

18) How can we render a pain HTML?

There is no need to "render" HTML with the res.render() function. If you have a specific file, you can use the res.sendFile() function, but you should use the express if you serve many assets from a directory.static() middleware function.

1) What is ExpressJs?

Express Js is a framework for node.js which is light-weight and fast. It is used to develop web and mobile applications.

2) What are the features of ExpressJs?

Following are the features of Expressjs:

- It allows middleware set up to respond HTTP Requests
- It defines routing table which is used to achieve action based on HTTP Method and URL.
- It allows dynamically generate the HTML Pages based on passing arguments to the templates.
- It follows MVC architecture for web application.

3) What is middleware in ExpressJs?

Middleware is a function that accesses the request object (req), response object (res), and access next middleware function in application's request-response cycle. The next middleware function is represented by variable called 'next'. **Syntax:**

```
function(req,res,next){ }
```

4) Can middleware is able for error handling? Explain error handling in ExpressJs.

Middleware is able to handle error handling in ExpressJs. This can be done by passing one extra error-handling (err) parameter in middleware function. **Syntax:**

```
function(err,req,res,next){ }
```

5) What is 'next' in ExpressJs?

'next' is a parameter which is passed in middleware function to pass control to next middleware.

```
function(req,res,next){ }
```

6) What is routing?

Routing define how an application responds over client request to a particular endpoint (URI). Endpoint is specially a path and any one of HTTP request methods (GET, POST, etc). Following structure is used for routing: **app.METHOD(PATH,HANDLER)**

Note: METHOD is Http request method (in lowercase),
PATH is uri on server, **HANDLER** is function name.

7) How to configure properties in ExpressJs application?

We can configure properties in two ways:

I.
. With **Process.env.**

- Create a file with extension '**.env**' inside the project folder.
- Add all the properties in '**.env**' file.
- In **server.js** any of properties can use:

```
var host=process.env.APP_HOST;

app.set('host',host);

logger.info(app.get('host');
```

I. With **RequireJs.**

```
Create a file 'config.json' inside 'config' folder of the project folder.

Add config properties in 'config.json' file.

Use require keyword to access the 'config.json' file.
var config =require('./config/config.json');
```

8) How debugging are done in ExpressJs?

Different operating system uses different commands:

```
Windows: set DEBUG = express:* & node index.js


Linux: $ DEBUG = express:* & node index.js
```

9) Define templating in ExpressJs?

Templating are powerful engine used for removing the cluttering of our server code with HTML. Some of the templates which are with ExpressJs are:

- Pug
- Mustache
- EJS

10) What is cookie and what it does?

Cookie is a data sent from server and stores on the client side. It keeps the information of user's actions. These are some following purpose of using cookie:

- Session management
- User tracking
- Personalization

11) How to use cookies in ExpressJs?

Cookies are used in ExpressJs by installing 'cookie-parser' middleware. To install cookie-parser we use command as:

```
npm install --save cookie-parser

To set new cookie in our application, we define a new route.


app.get('/',function(req, res){


res.cookie('name','express').send('cookie set')
});
```

12) How to delete cookie in ExpressJs?

The **clearCookie** method is used to delete cookies in ExpressJs.

```
clearCookie('cookie_name');
app.get('/clear_cookie_temp',function(req,res)){

      clearCookie('temp');

   res.send('cookie temp is cleared');

   });
```

13) How to handle 404 errors or redirect error to another page?

We use the following code in **server.js** file to redirect 404 errors to another page in ExpressJs.

```
app.use(function(req, res, next){

   res.status(404).json({errorCode: 404, errorMessage:
 requested resources not foundâ€•  });

});
```

14) What is scaffolding?

Scaffolding is a tool, which set up all required public directory, add middleware, create separate route file etc (set up skeleton for web application). So that we can directly get started building our application. **Yeoman** is scaffolding tool built for Node.js.

15) How to get the full url In ExpressJs?

```
var port = req.app.settings.port || cfg.port;


res.locals.requested_url = req.protocol + '://' + req.host  +

( port == 80 || port == 443 ? '' : ':'+port ) + req.path;
```

Q:- What is Express JS?

1. Express.js is a free open-source, flexible node.js web application framework.
2. It is designed for building backend web applications and APIs, etc.
3. It has been developed by TJ Holowaychuk in 2010 and written in JavaScript.

Express.js key features:

Following are the key features of express framework:

1. **Middlewares:** Express middleare are functions that access of - request (req), response (res) and next().
2. **Routing:** It is used to define routes in order to perform and handle HTTP requests/operations.
3. **Templates Engine:** It has SSR renders html template used to render the HTML on the browser.
4. **High Performance:** Express prepare a thin layer, therefore, the performance is adequate.
5. **Database Support:** Express supports RDBMS as well as NoSQL databases - MySQL, MongoDB, etc.
6. **MVC Support:** It is a web application which supports MVC architecture.
7. **ORM Support:** It support various ORM/ODM - Mongoose, Sequelize, etc.

Q:- What is Scaffolding in Express.js?

Scaffolding is creating the skeleton structure of application

There are 2 way to do this, by using:

1. Express Application Generator
2. Yeoman

1. Express Application Generator:

Use express-generator tool to quickly create an skeleton of your application.

```
npm install -g express-generator express myTestExpressApp
```

Th above command will create your project named - myTestExpressApp with followings.

1. **bin:**It is also called **www** is the main configuration file of the app.
2. **public:**It contains static assets - javascript, css, images, etc.
3. **routes:** It contains route files of the app.
4. **views:** It contains the view files (HTML) of the application.
5. **app.js:** It is the main file of the application which execute first.
6. **package.json:** It is the manifest file. It contains all metadata info of the project.

Install all the dependencies mentioned in the package.json file:

```
cd myTestExpressApp npm install
```

Q:- How to enable debugging in express app?

In different Operating Systems, we have following commands:

On Linux:

```
DEBUG=express:* node app.js
```

On Windows:

```
set DEBUG=express:* node app.js
```

Q:- Serving static files in Express.js?

```
app.use(express.static('public')) app.use('/static', express.static(path.join(__dirname, 'public')));
```

Q:- What is routing and how routing works in Express.js?

Routing refers to determining how an application responds to a request.

Route Syntax:

```
app.METHOD(PATH, HANDLER);
```

Where:

- **app** is an instance of **express**.
- **METHOD** is an HTTP request method (get, post, put, etc.).
- **PATH** is a path/endpoint on the server.
- **HANDLER** is a function executed when the route is matched.

#Example: A route with path / and get method.

```
app.get('/', function (req, res) { res.send('Express.js Interview Questions') })
```

Q:- How dynamic routing works in express.js?

When someone pass parameters in URL (i.e. Parametrized URL), this routing phenomenon is called dynamic routing.

```
var express = require('express'), app = express(); app.get('/article/:id', function(req ,
res){ res.render('article' + req.params.id); })
```

In above example: id is a parameters, which can be different for different request.

Q:- What is Middleware in express.js?

Middleware is a function that is invoked by the express routing layer before the final request processed.

Middleware functions can perform the following tasks:

1. Execute any code - validation, setting headers, etc.
2. You can make changes to the request (req) and response (res) objects.
3. You can also end the request-response cycle, if rquired.
4. You can call the next middleware function in the stack to proceed and process the final request.

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.

Types of Middleware:

1. Application-level middleware
2. Router-level middleware
3. Error-handling middleware
4. Built-in middleware
5. Third-party middleware

1. Application-level middleware:

This kind of middleware method is bind to the app Object using app.use() method. It applies on all routes.

```
//This middleware will execute for each route. app.use(function (req, res, next) {
console.log('Current Time:', Date.now()) next() })
```

2. Router-level middleware:

Router-level middleware binds to an specific instance of **express.Router()**

3. Built-in middleware:

Express has the following built-in middleware functions:

- **express.static** serves static assets such as HTML files, images, and so on.

- **express.json** parses incoming requests with JSON payloads.
- **express.urlencoded** parses incoming requests with URL-encoded payloads.

4. Third-party middleware:

There are a lots of third party middleware, such as

- Body-parser
- Cookie-parser
- Mongoose
- Sequelize
- Cors
- Express-validator

To handle HTTP POST request in express.js version 4 and above, you need to install a middleware module called body-parser, body-parser extract the entire body portion of an incoming request stream and exposes it on req.body, this middleware was a part of express.js earlier but now you have to install it separately.

These can be installed by using command:

```
>> npm install MODULE_NAME
```

And they can be loaded using requires and used later.

#Example:

```
var bodyParser = require('body-parser'); app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }))
```

Note: Multiple Middleware can be used as an array on a single route.

```
var middlewareArray = [middleware1, middleware2] app.get('/home',
middlewareArray, function (req, res, next) { //Code snippets })
```

Q:- Database integration in express.js?

Express.js supports of many RDBMS & NoSQL databases like

- MongoDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server
- SQLite

#Example: Install MongoDB

```
>>npm install mongodb var MongoClient = require('mongodb').MongoClient
MongoClient.connect('mongodb://localhost:27017/test_db', function (err, db) { if (err)
throw err db.collection('mammals').find().toArray(function (err, result) { if (err) throw
err console.log(result) }) })
```

Q:- Error handling in Express.js?

```
var express = require('express'), app = express(); app.use(function (err, req, res, next) { console.error(err.stack) // error first callback res.status(500).send('Something went wrong!') })

var express = require('express'), app = express(); app.use(function(req, res, next) { res.status(404).json({ errorCode: 404, errorMsg: "Not found!" }); });
```

Q:- How to allow CORS in Express.js (Node.js)? Explain with an Example?

Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources to be requested from another domain/server.

There are mainly three ways you can do this using

1. res.setHeader() - It allow to set only single header
2. res.header() OR res.set() - It allow to set multiple headers.
3. express cors module

1. Enable CORS for all resources using res.setHeader().

```
app.use(function(req, res, next) { // Website you wish to allow to connect res.setHeader("Access-Control-Allow-Origin", "*"); // Request methods you wish to allow res.setHeader( "Access-Control-Allow-Methods", "GET, POST, OPTIONS, PUT, PATCH, DELETE" ); // Request headers you wish to allow res.setHeader( "Access-Control-Allow-Headers", "X-Requested-With,content-type" ); // Set to true if you need the website to include cookies in the requests sent // to the API (e.g. in case you use sessions) res.setHeader("Access-Control-Allow-Credentials", true); // Pass to next layer of middleware next(); });
```

2. Enable CORS for all resources using res.header()

```
app.use(function(req, res, next) { res.header("Access-Control-Allow-Origin", "*"); res.header( "Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept" ); next(); });
```

3. Enable CORS for all resources using express CORS module.

If you are using **express.js** you can use **cors module**

3.1) Enable CORS Requests for All Servers.

```
var express = require("express"); var cors = require("cors"); var app = express(); app.use(cors()); app.get("/", function(req, res, next) { res.json({ msg: "This is CORS-enabled for all origins!" }); }); app.listen(80, function() { console.log("CORS-enabled web server listening on port 80"); });
```

3.2) Enable CORS Requests for Some Specific Servers

```
var express = require("express"); var cors = require("cors"); var app = express(); var whitelist = ["http://server1.com", "http://server2.com"]; var corsOptions = { origin: function(origin, callback) { if (whitelist.indexOf(origin) !== -1) { callback(null, true); } else { callback(new Error("Not allowed by CORS")); } } }; app.get("/",
cors(corsOptions), function(req, res, next) { res.json({ msg: "This is CORS-enabled for a whitelisted domain." }); }); app.listen(80, function() { console.log("CORS-enabled web server listening on port 80"); });
```

Q:- How to implement JWT authentication in Express app ? Explain with an Example?

- Create a folder called 'keys' inside project folder.
- Install some dependencies as following: npm install jsonwebtoken –-save
- Add the login router routes/index.js

```
router.post('/login, function(req, res) { // find the user User.findOne({ name: req.body.username }, function(err, res) { if (err) throw err; if (!res) { res.json({ success: false, message: Login failed.' }); } else if (res) { // check if password matches if (res.password != req.body.password) { res.json({ success: false, message: Login failed. Wrong password.' }); } else { var token = jwt.sign(res, app.get('superSecret'), { expiresInMinutes: 1600 }); // return the information including token as JSON res.json({ success: true, message: 'Valid token!', token: token }); } } }); });
```

- Use the token in application

```
jwt = require("express-jwt"); app.use(function(req, res, next) { var token = req.body.token || req.query.token || req.headers['x-access-token']; if (token) { jwt.verify(token, app.get('superSecret'), function(err, decoded) { if (err) { return res.json({ success: false, message: 'Invalid token.' }); } else { req.decoded = decoded; next(); } }); } else { return res.status(403).send({ success: false, message: 'No token given.' }); } });
```