1. What is node.js

Short Answer

Node.js is a Server side scripting which is used to build scalable programs. Its multiple advantages over other server side languages, the prominent being non-blocking I/O.

Long Answer

Node.js is a server-side technology that’s based on Google’s V8 JavaScript engine. It’s a highly scalable system that uses asynchronous, event-driven I/O (input/output), rather than threads or separate processes. It’s ideal for web applications that are frequently accessed but computationally simple.

If you’re using a traditional web server, such as Apache, each time a web resource is requested, Apache creates a separate thread or invokes a new process in order to handle the request. Even though Apache responds quickly to requests, and cleans up after the request has been satisfied, this approach can still tie up a lot of resources. A popular web application is going to have serious performance issues.

Node, on the other hand, doesn’t create a new thread or process for every request. Instead, it listens for specific events, and when the event happens, responds accordingly. Node doesn’t block any other request while waiting for the event functionality to complete, and events are handled—first come, first served—in a relatively uncomplicated event loop. Node applications are created with JavaScript (or an alternative language that compiles to JavaScript). The JavaScript is the same as you’d use in your client-side applications. However, unlike JavaScript in a browser, with Node you have to set up a development environment.

Node can be installed in a Unix/Linux, Mac OS, or Windows environment. This chapter will walk you through setting up a development environment for Node in Windows 7 and Linux (Ubuntu). Installation on a Mac should be similar to installation on Linux. I’ll also cover any requirements or preparation you need to take before installing the application. Once your development environment is operational, I’ll demonstrate a basic Node application and walk you through the important bit—the event loop I mentioned earlier.

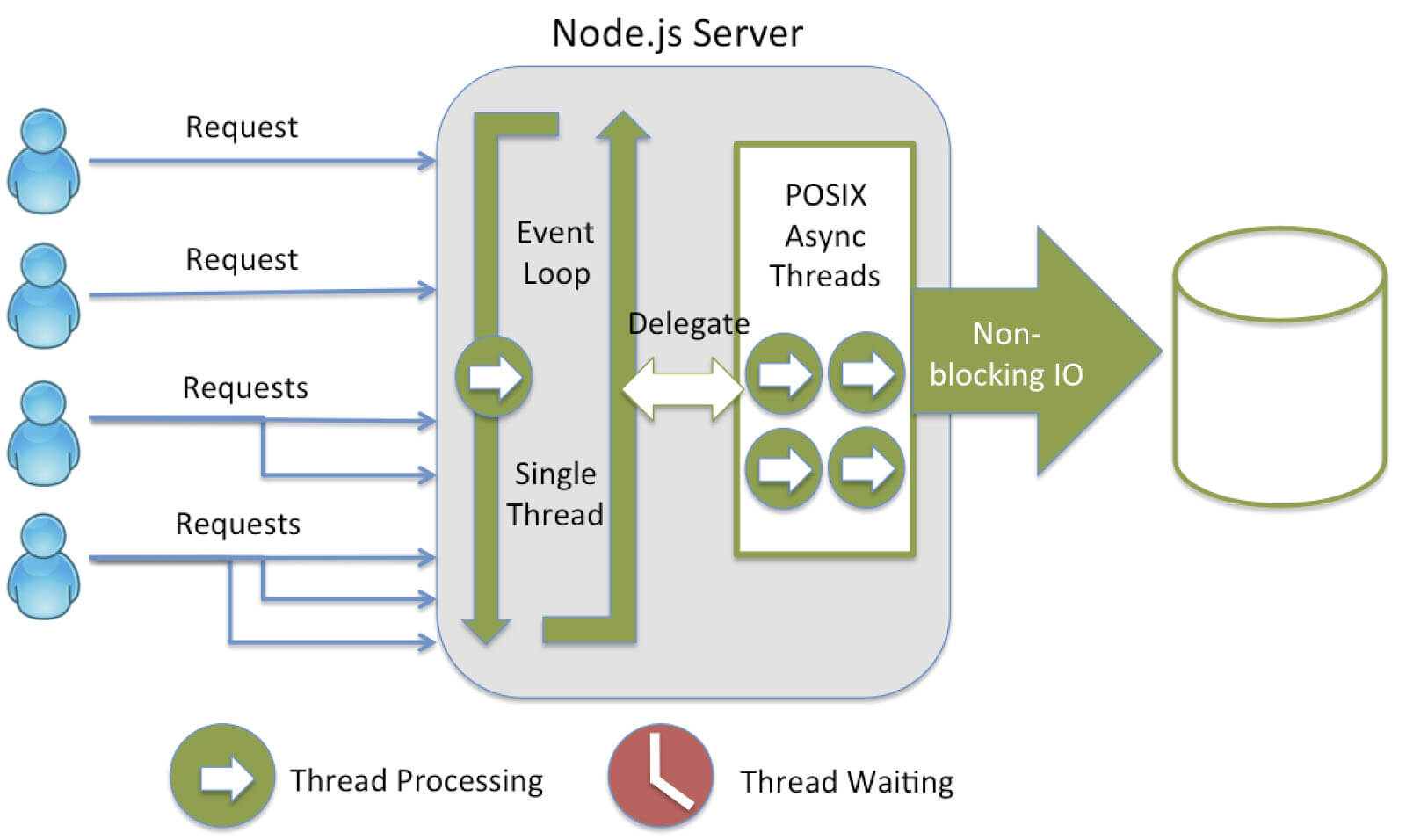
2. How node.js works

Short Answer

Node.js works on a v8 environment, it is a virtual machine that utilizes JavaScript as its scripting language and achieves high output via non-blocking I/O and single threaded event loop.

Long Answer

NodeJS is built on top of V8, Google’s Javascript engine that also powers the Chrome browser, which sported the fastest Javascript execution speeds and pretty much put Javascript performance right up in front when it came to comparing browsers, turning speed into a major marketing for browsers.



3. What do you mean by the term I/O ?

I/O is the shorthand for input and output, and it will access anything outside of your application. It will be loaded into the machine memory to run the program, once the application is started.

4. What does event-driven programming mean?

In computer programming, event driven programming is a programming paradigm in which the flow of the program is determined by events like messages from other programs or threads.

5. Where can we use node.js?

Node.js shines in real-time web applications employing push technology over web sockets. It can be used for the following purposes: a) Web applications (especially real-time web apps) b) Network applications c) Distributed systems d) General purpose applications

6. What is the advantage of using node.js?

It provides an easy way to build scalable network programs a) Easy to install and run locally b) Generally fast c) Great concurrency d) Asynchronous everything e) Almost never blocks f) Unified programming language and data type

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

The two types of API functions in Node.js are: a) Asynchronous, non-blocking functions b) Synchronous, blocking functions

8. What is control flow function?

A generic piece of code which runs in between several asynchronous function calls is known as control flow function.

9. Explain the steps how “Control Flow” controls the functions calls?

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

10. Why Node.js is single threaded?

For async processing, Node.js was created explicitly as an experiment. It is believed that more performance and scalability can be achieved by doing async processing on a single thread under typical web loads than the typical thread based implementation.

11. Does node run on windows?

A) No

B) yes

check answer

12. Can you access DOM in node?

No, you cannot access DOM in node.

13. Using the event loop what are the tasks that should be done asynchronously?

a. I/O operations b. Heavy computation c. Anything requiring blocking

14. Why node.js is quickly gaining attention from JAVA programmers?

Node.js is quickly gaining attention as it is a loop based server for JavaScript. Node.js gives user the ability to write the JavaScript on the server, which has access to things like HTTP stack, file I/O, TCP and databases.

15. What are the two arguments that async.queue takes?

The two arguments that async.queue takes a) Task function b) Concurrency value

16. What is an event loop in Node.js ?

To process and handle external events and to convert them into callback invocations an event loop is used. So, at I/O calls, node.js can switch from one request to another .

17. Mention the steps by which you can async in Node.js?

By following steps you can async Node.js : First class functions -> Function composition -> Callback Counters -> Event loops

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

Pros:

a) If your application does not have any CPU intensive computation, you can build it in Javascript top to bottom, even down to the database level if you use JSON storage object DB like MongoDB.

b) Crawlers receive a full-rendered HTML response, which is far more SEO friendly rather than a single page application or a websockets app run on top of Node.js.

Cons:

a) Any intensive CPU computation will block node.js responsiveness, so a threaded platform is a better approach.

b) Using relational database with Node.js is considered less favorable.

19. How Node.js overcomes the problem of blocking of I/O operations?

Node.js solves this problem by putting the event based model at its core, using an event loop instead of threads.

20. What is the difference between Node.js vs Ajax?

The difference between Node.js and Ajax is that, Ajax (short for Asynchronous Javascript and XML) is a client side technology, often used for updating the contents of the page without refreshing it. While,Node.js is Server Side Javascript, used for developing server software. Node.js does not execute in the browser but by the server.

60+ Node.JS interview questions and best answers for freshers and 2-5 year experienced web developers.

1) What is Node.js?

Node.js is a very powerful JavaScript based platform or framework which is built on Google Chrome's JavaScript V8 Engine.

2) Why to use Node.js?

It is used to develop I/O intensive web applications like video streaming sites, single page applications (SPA) and other web applications. Node.js is open source and used by thousands of developers around the world.

3) Who developed Node.js?

Node.js was developed in 2009 by Ryan Dahl.

4) What are the features of Node.js?

Below are the features of Node.js –

Very Fast

Event driven and Asynchronous

Single Threaded but highly Scalable

5) Explain REPL in Node.js?

REPL stands for Read Eval Print Loop. Node.js comes with bundled REPL environment which performs the following desired tasks –

Eval

Print

Loop

Read

6) Explain variables in Node.js?

Variables are used to store values and print later like any conventional scripts. If “var” keyword is used then value is stored in variable. You can print the value in the variable using - console.log().

Eg:

$ node

> a = 30

30

> var b = 50

undefined

> a + b

80

> console.log("Hi")

Hi

undefined

7) What is the latest version of Node.js available?

Latest version of Node.js is - v0.10.36.

8) List out some REPL commands in Node.js?

Below are the list of REPL commands –

Ctrl + c - For terminating the current command.

Ctrl + c twice – For terminating REPL.

Ctrl + d - For terminating REPL.

Tab Keys - list of all the current commands.

.break - exit from multiline expression.

.save with filename - save REPL session to a file.

9) Mention the command to stop REPL in Node.js?

Command - ctrl + c twice is used to stop REPL.

10) Explain NPM in Node.js?

NPM stands for Node Package Manager (npm) and there are two functionalities which NPM takes care of mainly and they are –

Online repositories for node.js modules or packages, which can be searched on search.nodejs.org

Dependency Management, Version Management and command line utility for installing Node.js packages.

11) Mention command to verify the NPM version in Node.js?

Below command can be used to verify the NPM version –

$ npm --version

12) How you can update NPM to new version in Node.js?

Below commands can be used for updating NPM to new version –

$ sudo npm install npm -g

/usr/bin/npm -> /usr/lib/node\_modules/npm/bin/npm-cli.js

npm@2.7.1 /usr/lib/node\_modules/npm

13) Explain callback in Node.js?

Callback is called once the asynchronous operation has been completed. Node.js heavily uses callbacks and all API’s of Node.js are written to support callbacks.

14) How Node.js can be made more scalable?

Node.js works good for I/O bound and not CPU bound work. For instance if there is a function to read a file, file reading will be started during that instruction and then it moves onto next instruction and once the I/O is done or completed it will call the callback function. So there will not be any blocking.

15) Explain global installation of dependencies?

Globally installed dependencies or packages are stored in <user-directory>/npm directory and these dependencies can be used in Command Line Interface function of any node.js.

16) Explain local installation of dependencies?

By default npm will install the 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. “require ()” is used to access the locally deployed packages.

17) Explain Package.JSON?

This will be present in the root directory of any Node module/application and will be used to define the properties of a package.

18) Explain “Callback hell”?

“Callback hell” will be referred to heavily nested callbacks which has become unreadable or unwieldly.

19) What are “Streams” in Node.JS?

“Streams” are objects which will let you read the data from source and write data to destination as a continuous process.

20) What you mean by chaining in Node.JS?

It’s a mechanism in which output of one stream will be connected to another stream and thus creating a chain of multiple stream operations.

21) Explain Child process module?

Child process module has following three major ways to create child processes –

spawn - child\_process.spawn launches a new process with a given command.

exec - child\_process.exec method runs a command in a shell/console and buffers the output.

fork - The child\_process.fork method is a special case of the spawn() to create child processes.

22) Why to use exec method for Child process module?

“exec” method runs a command in a shell and buffers the output. Below is the command –

child\_process.exec(command[, options], callback)

23) List out the parameters passed for Child process module?

Below are the list of parameters passed for Child Process Module –

child\_process.exec(command[, options], callback)

command - This is the command to run with space-separated arguments.

options – This is an object array which comprises one or more following options –

cwd

uid

gid

killSignal

maxBuffer

encoding

env

shell

timeout

callback – This is the function which is gets 2 arguments – stdout, stderr and error.

24) What is the use of method – “spawn()”?

This method is used to launch a new process with the given commands. Below is the method signature –

child\_process.spawn(command[, args][, options])

25) What is the use of method – “fork()”?

This method is a special case for method- “spawn()” for creating node processes. The method signature –

child\_process.fork(modulePath[, args][, options])

26) Explain Piping Stream?

This is a mechanism of connecting one stream to other and this is basically used for getting the data from one stream and pass the output of this to other stream.

27) What would be the limit for Piping Stream?

There will not be any limit for piping stream.

28) Explain FS module ?

Here FS stands for “File System” and fs module is used for File I/O. FS module can be imported in the following way –

var test = require("fs")

29) Explain “Console” in Node.JS?

“Console” is a global object and will be used for printing to stderr and stdout and this will be used in synchronous manner in case of destination is either file or terminal or else it is used in asynchronous manner when it is a pipe.

30) Explain – “console.log([data][, ...])” statement in Node.JS?

This statement is used for printing to “stdout” with newline and this function takes multiple arguments as “printf()”.

31) What you mean by “process”?

“process” is a global object and will be used to represent a node process.

32) Explain exit codes in Node.JS? List out some exit codes?

Exit code will be used when the process needs to be ended with specified code. Below are the list of exit codes in Node.JS –

Fatal Error

Non-function Internal Exception Handler

Internal JavaScript Parse Error

Uncaught Fatal Exception

Unused

Internal JavaScript Evaluation Failure

Internal Exception Handler Run-Time Failure

33) List out the properties of process?

Below are the useful properties of process –

Platform

Stdin

Stdout

Stderr

execPath

mainModule

execArgv

config

arch

title

version

argv

env

exitCode

34) Define OS module?

OS module is used for some basic operating system related utility functions. Below is the syntax for importing OS module –

var MyopSystem = require("os")

35) What is the property of OS module?

os.EOL – Constant for defining appropriate end of line marker for OS.

36) Explain “Path” module in Node.JS?

“Path” module will be used for transforming and handling file paths. Below is the syntax of path module –

var mypath = require("path")

37) Explain “Net” module in Node.JS?

“Net” module is being used for creating both clients and servers. It will provide asynchronous network wrapper. Below is the syntax of Net module –

var mynet = require("net")

38) List out the differences between AngularJS and NodeJS?

AngularJS is a web application development framework. It’s a JavaScript and it is different from other web app frameworks written in JavaScript like jQuery. NodeJS is a runtime environment used for building server-side applications while AngularJS is a JavaScript framework mainly useful in building/developing client-side part of applications which run inside a web browser.

39) NodeJS is client side server side language?

NodeJS is a runtime system, which is used for creating server-side applications.

40) What are the advantages of NodeJS?

Below are the list of advantages of NodeJS –

Javascript – It’s a javascript which can be used on frontend and backend.

Community Driven - NodeJS has great open source community which has developed many excellent modules for NodeJS to add additional capabilities to NodeJS applications.

41) In which scenarios NodeJS works well?

NodeJS is not appropriate to use in scenarios where single-threaded calculations are going to be the holdup.

42) What you mean by JSON?

JavaScript Object Notation (JSON) is a practical, compound, widely popular data exchange format. This will enable JavaScript developers to quickly construct APIs.

43) Explain “Stub”?

Stub is a small program, which substitutes for a longer program, possibly to be loaded later and that is located remotely. Stubs are functions/programs that simulate the behaviors of components/modules.

44) List out all Node.JS versions available?

Below are the list of all NodsJS versions supported in operating systems –

OperatingSystem Node.js version

Windows node-v0.12.0-x64.msi

Linux node-v0.12.0-linux-x86.tar.gz

Mac node-v0.12.0-darwin-x86.tar.gz

SunOS node-v0.12.0-sunos-x86.tar.gz

45) Explain “Buffer class” in Node.JS?

It is a global class which can be accessed in an application without importing buffer modules.

46) How we can convert Buffer to JSON?

The syntax to convert Buffer to JSON is as shown beow

buffer.toJSON()

47) How to concatenate buffers in NodeJS?

The syntax to concatenate buffers in NodeJS is

var MyConctBuffer = Buffer.concat([myBuffer1, myBuffer2]);

48) How to compare buffers in NodeJS?

To compare buffers in NodeJS, use following code –

Mybuffer1.compare(Mybuffer2);

49) How to copy buffers in NodeJS?

Below is the syntax to copy buffers in NodeJS –

buffer.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd])

50) What are the differences between “readUIntBE” and “writeIntBE” in Node.JS?

readUIntBE - It’s a generalized version of all numeric read methods, which supports up to 48 bits accuracy. Setting noAssert to “true” to skip the validation.

writeIntBE - This will write the value to the buffer at the specified byteLength and offset and it supports upto 48 bits of accuracy.

51) Why to use “\_\_filename” in Node.JS?

“\_\_filename” is used to represent the filename of the code which is being executed. It used to resolve the absolute path of file. Below is the sample code for the same –

Console.log(\_\_filename);

52) Why to use “SetTimeout” in Node.JS?

This is the global function and it is used to run the callback after some milliseconds.

Syntax of this method –

setTimeout(callbackmethod, millisecs)

53) Why to use “ClearTimeout” in Node.JS?

This is the global function and it is used to stop a timer which was created during “settimeout()”.

54) Explain Web Server?

It is a software app which will handle the HTTP requests by client (eg: browser) and will return web pages to client as a response. Most of web server supports – server side scripts using scripting languages. Example of web server is Apache, which is mostly used webserver.

55) List out the layers involved in Web App Architechure?

Below are the layers used in Web Apps –

Client - Which makes HTTP request to the server. Eg: Browsers.

Server – This layer is used to intercept the requests from client.

Business – It will have application server utilized by web servers for processing.

Data – This layer will have databases mainly or any source of data.

56) Explain “Event Emitter” in Node.JS?

It is a part of Events module. When instance of EventEmitter faces any error, it will emit an 'error' event. “Event Emitters” provides multiple properties like – “emit” and “on”.

“on” property is used for binding the function with event.

“emit” property is used for firing an event.

57) Explain “NewListener” in Node.JS?

This event is being emitted whenever any listener is added. So when event is triggered the listener may not have been removed from listener array for the event.

58) Why to use Net.socket in Node.JS?

This object is an abstraction of a local socket or TCP. net.Socket instances implement a duplex Stream interface. These can be created by the user and used as a client (with connect() function) or they can be created by Node and can be passed to the user through the 'connection' event of a server.

59) Which events are emitted by Net.socket?

Below are the list of events emitted by Net.socket –

Connect

Lookup

End

Data

Close

Drain

Timeout

Error

60) Explain “DNS module” in Node.JS?

This module is used for DNS lookup and to use underlying OS name resolution. This used to provide asynchronous network wrapper. DNS module can be imported like –

var mydns = require("dns")

61) Explain binding in domain module in Node.JS?

Below are the bindings in domain modules –

External Binding

Internal Binding

62) Explain RESTful Web Service?

Web services which uses REST architecture will be known as RESTful Web Services. These web services uses HTTP protocol and HTTP methods.

63) How to truncate the file in Node.JS?

Below command can be used for truncating the file –

fs.ftruncate(fd, len, callback)

Consider the following JavaScript code:

console.log("first");

setTimeout(function() {

console.log("second");

}, 0);

console.log("third");

The output will be:

first

third

second

Assuming that this is the desired behavior, and that we are using Node.js version 0.10 or higher, how else might we write this code?

Hide answer

Node.js version 0.10 introduced setImmediate, which is equivalent to setTimeout(fn, 0), but with some slight advantages.

**setTimeout(fn, delay)** calls the given callback fn after the given delay has ellapsed (in milliseconds). However, the callback is not executed immediately at this time, but added to the function queue so that it is executed **as soon as possible**, after all the currently executing and currently queued event handlers have completed. Setting the delay to 0 adds the callback to the queue immediately so that it is executed as soon as all currently-queued functions are finished.

**setImmediate(fn)** achieves the same effect, except that it doesn’t use the queue of functions. Instead, it checks the queue of I/O event handlers. If all I/O events in the current snapshot are processed, it executes the callback. It queues them immediately after the last I/O handler somewhat like process.nextTick. This is faster than setTimeout(fn, 0).

So, the above code can be written in Node as:

console.log("first");

setImmediate(function(){

console.log("second");

});

console.log("third");

[Comment](https://www.toptal.com/nodejs/interview-questions)

What is “callback hell” and how can it be avoided?

Hide answer

“Callback hell” refers to heavily nested callbacks that have become unweildy or unreadable.

An example of heavily nested code is below:

query("SELECT clientId FROM clients WHERE clientName='picanteverde';", function(id){

query("SELECT \* FROM transactions WHERE clientId=" + id, function(transactions){

transactions.each(function(transac){

query("UPDATE transactions SET value = " + (transac.value\*0.1) + " WHERE id=" + transac.id, function(error){

if(!error){

console.log("success!!");

}else{

console.log("error");

}

});

});

});

});

The primary method to fix callback hell is usually referred to as **modularization**. The callbacks are broken out into independent functions which can be called with some parameters. So the first level of improvement might be:

var logError = function(error){

if(!error){

console.log("success!!");

}else{

console.log("error");

}

},

updateTransaction = function(t){

query("UPDATE transactions SET value = " + (t.value\*0.1) + " WHERE id=" + t.id, logError);

},

handleTransactions = function(transactions){

transactions.each(updateTransaction);

},

handleClient = function(id){

query("SELECT \* FROM transactions WHERE clientId=" + id, handleTransactions);

};

query("SELECT clientId FROM clients WHERE clientName='picanteverde';",handleClient);

Even though this code is much easier to read, and we created some functions that we can even reuse later, in some cases it may be appropriate to use a more robust solution in the form of **promises**. Promises allow additional desirable behavior such as error propogation and chaining. Node.js doesn’t include much core support for promises, so one of the popular promise libraries should be used. One of the most popular is the [Q promise library](https://www.npmjs.org/package/q).

More information about promises and how they work can be found [here](http://www.html5rocks.com/en/tutorials/es6/promises/).

Additionally, a more supercharged solution to callback hell is provided by **generators**, as these can resolve execution dependency between different callbacks. However, generators are much more advanced and it might be overkill to use them for this purpose. To read more about generators you can start with [this post](http://strongloop.com/strongblog/how-to-generators-node-js-yield-use-cases/).

[Comment](https://www.toptal.com/nodejs/interview-questions)

How does Node.js handle child threads?

Hide answer

Node.js, in its essence, is a **single thread** process. It does not expose child threads and thread management methods to the developer. Technically, Node.js *does* spawn child threads for certain tasks such as asynchronous I/O, but these run behind the scenes and do not execute any application JavaScript code, nor block the main event loop.

If threading support is desired in a Node.js application, there are tools available to enable it, such as the [ChildProcess](http://nodejs.org/api/child_process.html" \t "_blank) module.

[Comment](https://www.toptal.com/nodejs/interview-questions)

**Find top Node.js talent today.**Toptal can match you with the best developers to finish your project.

[**HIRE TOPTAL’S NODE.JS DEVELOPERS**](https://www.toptal.com/nodejs)

What is the preferred method of resolving unhandled exceptions in Node.js?

Hide answer

Unhandled exceptions in Node.js can be caught at the Process level by attaching a handler for uncaughtException event.

process.on('uncaughtException', function(err) {

console.log('Caught exception: ' + err);

});

However, uncaughtException is a very crude mechanism for exception handling and may be removed from Node.js in the future. An exception that has bubbled all the way up to the Process level means that your application, and Node.js may be in an undefined state, and the only sensible approach would be to restart everything.

The preferred way is to add another layer between your application and the Node.js process which is called the [domain](http://nodejs.org/api/domain.html).

Domains provide a way to handle multiple different I/O operations as a single group. So, by having your application, or part of it, running in a separate domain, you can safely handle exceptions at the domain level, before they reach the Process level.

[Comment](https://www.toptal.com/nodejs/interview-questions)

How does Node.js support multi-processor platforms, and does it fully utilize all processor resources?

Hide answer

Since Node.js is by default a **single thread** application, it will run on a single processor core and will not take full advantage of multiple core resources. However, Node.js provides support for deployment on multiple-core systems, to take greater advantage of the hardware. The [Cluster](http://nodejs.org/api/cluster.html) module is one of the core Node.js modules and it allows running multiple Node.js worker processes that will share the same port.

[Comment](https://www.toptal.com/nodejs/interview-questions)

What is typically the first argument passed to a Node.js callback handler?

Hide answer

Node.js core modules, as well as most of the community-published ones, follow a pattern whereby the first argument to any callback handler is an optional error object. If there is no error, the argument will be null or undefined.

A typical callback handler could therefore perform error handling as follows:

function callback(err, results) {

// usually we'll check for the error before handling results

if(err) {

// handle error somehow and return

}

// no error, perform standard callback handling

}

[Comment](https://www.toptal.com/nodejs/interview-questions)

Consider following code snippet:

{

console.time("loop");

for (var i = 0; i < 1000000; i += 1){

// Do nothing

}

console.timeEnd("loop");

}

The time required to run this code in Google Chrome is considerably more than the time required to run it in [Node.js](http://nodejs.org/). Explain why this is so, even though both use the v8 JavaScript Engine.

Hide answer

Within a web browser such as Chrome, declaring the variable i outside of any function’s scope makes it global and therefore binds it as a property of the window object. As a result, running this code in a web browser requires repeatedly resolving the property i within the heavily populated window namespace in each iteration of the for loop.

In Node.js, however, declaring any variable outside of any function’s scope binds it only to the module’s own scope (not the window object) which therefore makes it much easier and faster to resolve.

**Node.js Interview Questions for 2017**

* What is an error-first callback?
* How can you avoid callback hells?
* What are Promises?
* What tools can be used to assure consistent style? Why is it important?
* When should you npm and when yarn?
* What's a stub? Name a use case!
* What's a test pyramid? Give an example!
* What's your favorite HTTP framework and why?
* How can you secure your HTTP cookies against XSS attacks?
* How can you make sure your dependencies are safe?

**The Answers**

**What is an error-first callback?**

Error-first callbacks are used to pass errors and data as well. You have to pass the error as the first parameter, and it has to be checked to see if something went wrong. Additional arguments are used to pass data.

fs.readFile(filePath, **function**(err, data) {

**if** (err) {

*// handle the error, the return is important here*

*// so execution stops here*

**return** console.log(err)

}

*// use the data object*

console.log(data)

})

**How can you avoid callback hells?**

There are lots of ways to solve the issue of callback hells:

* **modularization**: break callbacks into independent functions
* use a **control flow library**, like [async](https://www.npmjs.com/package/async)
* use **generators with Promises**
* use **async/await** *(note that it is only available in the latest v7 release and not in the LTS version -*[*you can read our experimental async/await how-to here*](https://blog.risingstack.com/async-await-node-js-7-nightly/)*)*

[**Q: How to avoid callback hells? A: modularization, control flow libraries, generators with promises, async/await**](https://twitter.com/share?text=Q%3AHow%20to%20avoid%20callback%20hells%3F%20A%3A%20modularization%2C%20control%20flow%20libraries%2C%20generators%20with%20promises%2C%20async%2Fawait;url=https://blog.risingstack.com/node-js-interview-questions-and-answers-2017)

[CLICK TO TWEET](https://twitter.com/share?text=Q%3A%20How%20to%20avoid%20callback%20hells%3F%20A%3A%20modularization%2C%20control%20flow%20libraries%2C%20generators%20with%20promises%2C%20async%2Fawait;url=https://blog.risingstack.com/node-js-interview-questions-and-answers-2017)

**What are Promises?**

Promises are a concurrency primitive, first described in the 80s. Now they are part of most modern programming languages to make your life easier. Promises can help you better handle async operations.

An example can be the following snippet, which after 100ms prints out the result string to the standard output. Also, note the catch, which can be used for error handling. Promises are chainable.

**new** Promise((resolve, reject) => {

setTimeout(() => {

resolve('result')

}, 100)

})

.then(console.log)

.catch(console.error)

**What tools can be used to assure consistent style? Why is it important?**

When working in a team, consistent style is important, so team members can modify more projects easily, without having to get used to a new style each time.

Also, it can help eliminate programming issues using static analysis.

Tools that can help:

* [ESLint](http://eslint.org/)
* [Standard](http://standardjs.com/)

If you’d like to be even more confident, I suggest you to learn and embrace the [JavaScript Clean Coding](https://blog.risingstack.com/javascript-clean-coding-best-practices-node-js-at-scale/) principles as well!

**What's a stub? Name a use case!**

Stubs are functions/programs that simulate the behaviors of components/modules. Stubs provide canned answers to function calls made during test cases.

An example can be writing a file, without actually doing so.

**var** fs = require('fs')

**var** writeFileStub = sinon.stub(fs, 'writeFile', **function** (path, data, cb) {

**return** cb(null)

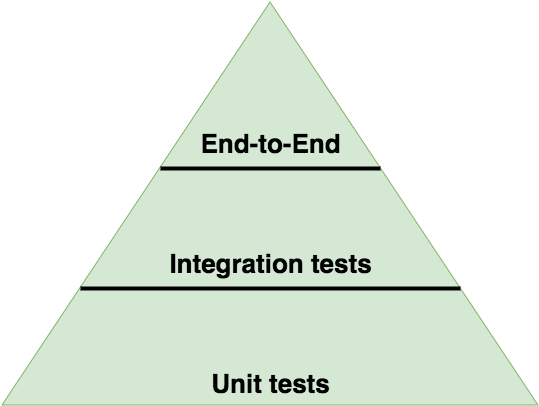
})

expect(writeFileStub).to.be.called

writeFileStub.restore()

**What's a test pyramid? Give an example!**

A test pyramid describes the ratio of how many unit tests, integration tests and end-to-end test you should write.



An example for an HTTP API may look like this:

* lots of low-level unit tests for models *(dependencies****are stubbed****)*,
* fewer integration tests, where you check how your models interact with each other *(dependencies****are not stubbed****)*,
* less end-to-end tests, where you call your actual endpoints *(dependencies****are not stubbed****)*.

**What's your favorite HTTP framework and why?**

There is no right answer for this. The goal here is to understand how deeply one knows the framework she/he uses. Tell what are the pros and cons of picking that framework.

**When are background/worker processes useful? How can you handle worker tasks?**

Worker processes are extremely useful if you'd like to do data processing in the background, like sending out emails or processing images.

**How can you secure your HTTP cookies against XSS attacks?**

XSS occurs when the attacker injects executable JavaScript code into the HTML response.

To mitigate these attacks, you have to set flags on the set-cookieHTTP header:

* **HttpOnly** - this attribute is used to help prevent attacks such as cross-site scripting since it does not allow the cookie to be accessed via JavaScript.
* **secure** - this attribute tells the browser to only send the cookie if the request is being sent over HTTPS.

So it would look something like this: Set-Cookie: sid=<cookie-value>; HttpOnly. If you are using Express, with [express-cookie session](https://github.com/expressjs/cookie-session#cookie-options), it is working by default.

**How can you make sure your dependencies are safe?**

When writing Node.js applications, **ending up with hundreds or even thousands of dependencies can easily happen**.   
For example, if you depend on Express, you depend on [27 other modules](https://github.com/expressjs/express/blob/master/package.json#L29) directly, and of course on those dependencies' as well, so manually checking all of them is not an option!

The only option is to automate the update / security audit of your dependencies. For that there are free and paid options:

* npm outdated
* [Trace by RisingStack](https://trace.risingstack.com/)
* [NSP](https://nodesecurity.io/)
* [GreenKeeper](https://greenkeeper.io/)
* [Snyk](https://snyk.io/)

**Node.js Interview Puzzles**

The following part of the article is useful if you’d like to prepare for an interview that involves puzzles, or tricky questions.

**What's wrong with the code snippet?**

**new** Promise((resolve, reject) => {

**throw** **new** Error('error')

}).then(console.log)

**The Solution**

As there is no catch after the then. This way the error will be a silent one, there will be no indication of an error thrown.

To fix it, you can do the following:

**new** Promise((resolve, reject) => {

**throw** **new** Error('error')

}).then(console.log).catch(console.error)

If you have to debug a huge codebase, and you don't know which Promise can potentially hide an issue, you can use the unhandledRejection hook. It will print out all unhandled Promise rejections.

process.on('unhandledRejection', (err) => {

console.log(err)

})

**What's wrong with the following code snippet?**

**function** **checkApiKey** (apiKeyFromDb, apiKeyReceived) {

**if** (apiKeyFromDb === apiKeyReceived) {

**return** true

}

**return** false

}

**The Solution**

When you compare security credentials it is crucial that you don't leak any information, so you have to make sure that you compare them in fixed time. If you fail to do so, your application will be vulnerable to [timing attacks](https://en.wikipedia.org/wiki/Timing_attack).

But why does it work like that?

**V8, the JavaScript engine used by Node.js, tries to optimize the code you run from a performance point of view.** It starts comparing the strings character by character, and once a mismatch is found, it stops the comparison operation. **So the longer the attacker has right from the password, the more time it takes.**

To solve this issue, you can use the npm module called [cryptiles](https://www.npmjs.com/package/cryptiles).

**function** **checkApiKey** (apiKeyFromDb, apiKeyReceived) {

**return** cryptiles.fixedTimeComparison(apiKeyFromDb, apiKeyReceived)

}

**What's the output of following code snippet?**

Promise.resolve(1)

.then((x) => x + 1)

.then((x) => { **throw** **new** Error('My Error') })

.catch(() => 1)

.then((x) => x + 1)

.then((x) => console.log(x))

.catch(console.error)

**The Answer**

The short answer is 2 - however with this question **I'd recommend asking the candidates to explain what will happen line-by-line to understand how they think**. It should be something like this:

1. A new Promise is created, that will resolve to 1.
2. The resolved value is incremented with 1 (so it is 2 now), and returned instantly.
3. The resolved value is discarded, and an error is thrown.
4. The error is discarded, and a new value (1) is returned.
5. The execution did not stop after the catch, but before the exception was handled, it continued, and a new, incremented value (2) is returned.
6. The value is printed to the standard output.
7. This line won't run, as there was no exception.