Node.js Interview Questions and Answers !?

by **Gergely Nemeth** (@nthgergo) – Co-founder of RisingStack

Two years ago we published our first article on common Node.js Interview Questions and Answers. Since then a lot of things improved in the JavaScript and Node.js ecosystem, so it was time to update it.

Important Disclaimers

It is never a good practice to judge someone just by questions like these, but **these can give you an overview of the person's experience in Node.js.**

But obviously, these questions do not give you the big picture of someone's mindset and thinking.

I think that a real-life problem can show a lot more of a candidate's knowledge - **so we encourage you to do pair programming with the developers you are going to hire**.

Finally and most importantly: we are all humans, so make your hiring process as welcoming as possible. These questions are not meant to be used as *"Questions & Answers"* but just to drive the conversation.

Node.js Interview Questions

- 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

the LTS version – you can read our experimental async/await how-to here)

Q: How to avoid callback hells? A: modularization, control flow libraries, generators with promises, async/await

CLICK TO TWEET

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.

- ESLint
- Standard

If you'd like to be even more confident, I suggest you to learn and embrace the JavaScript Clean Coding 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.

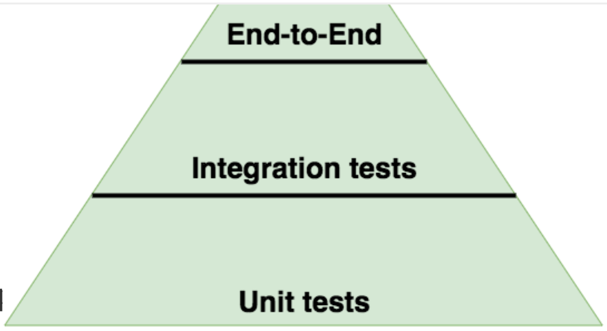
```
var fs = require('fs')

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.

There are lots of options for this like RabbitMQ or Kafka.

How can you secure your HTTP cookies against XSS attacks?

To mitigate these attacks, you have to set flags on the `set-cookie` HTTP 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, 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 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
- NSP
- GreenKeeper
- Snyk

Node.js Interview Puzzles

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
}
```

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.

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.

```
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:

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.

A day may work better than questions

Spending at least half a day with your possible next hire is worth more than a thousand of these questions.

Once you do that, you will better understand if the candidate is a good cultural fit for the company and has the right skill set for the job.

Do you miss anything? Let us know!

What was the craziest interview question you had to answer? What's your favorite question / puzzle to ask? Let us know in the comments! :)

TAGS:

Node.js Tutorials for Beginners Learn Node.js

Read more from us:

Hire RisingStack

Outsourced Development

24/7 Node.js Support

Training (fullstack & devops)

Get early access to our posts

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Hi

Outsourced Development

24/7 Node.js Support

Training (fullstack & devops)

Yash Thakkar • 2 years ago

I have a question/confusion.
In JS we follow error-first callback, then why in promise, first func is resolve? why not reject?

73 • Reply • Share ›

Simon S → Yash Thakkar • 8 months ago

The question doesn't make much sense. We technically don't have "error-first" callbacks. You can rearrange the "if-then" in a callback so that the non-exception code handles first. And as far as "error" being the 1st parameter, would your code be any different if the "error" was the 2nd parameter in the signature? (Hint: no)

As for promises, if 'catch' came before 'then', then I think it would block being able to use one 'catch' for 'many' promises. I haven't verified that.

• Reply • Share ›

Gergely Németh → Yash Thakkar • 2 years ago

It is just how the promise standard is created - I know it is a bit confusing, but for error-first callbacks you have to check the first parameter for errors, while when creating a new Promise, the first function is the resolve, the second is the reject

• Reply • Share ›

Kishan Kumar → Yash Thakkar • 9 months ago

Because sometimes it happens that if reject is the first func. it might catch the result also. Think of it like a SWITCH statements where we have default at the last and specific ones higher in that order.

You can find some more questions and answers here - https://asafdav2.github.io/...

20 ^ | v • Reply • Share ›

Liran Tal • 2 years ago

I liked most the security-oriented questions. It's easy to give in to the notion that security is someone else's job when it's really present in every aspect.

3 ^ | v • Reply • Share ›

billobeng • 2 years ago

i can't find the info on yarn vs npm too ?

3 ^ | v • Reply • Share ›

vitvad ➦ billobeng • 2 years ago

I have only one thought, about shrinkwrap, but as I heard at the end yarn has same problems with shrinkwrap as npm. So IMHO answer will be - do not use yarn at all.
@Gergely Németh , could you comment this ?

3 ^ | v • Reply • Share ›

TonyRedondo • 5 months ago

The question about timing attacks is not very useful on its current phrasing.

It is pretty much impossible to find out an API key or user password in a real life network. The function that would do the comparison would just be a function on a call stack of hundred of functions. On top of that add the TLS encryption time of all HTTP traffic. Then the time the POST response would take to reach the client, which is a very unreliable operation. And finally you would have to perform this operation for thousands of times (making the assumption the response times are always accurately correlated with the times of the string comparing function).

A better question related to Node.js security would be to ask about server-attack mitigation techniques. Rate limits (https://www.nginx.com/blog/..., DDoS protection (https://aws.amazon.com/shield) or techniques to prevent SQL injections would be more real-scenario

Good list of questions by the way.

1 • Reply • Share ›

Viktor Molokostov • 5 months ago

How come that the example for the "background processes" question is RabbitMQ/Kafka? Those are message queues, they don't do any processing. Yes, they are obviously separate processes, in most cases even executed on a different (virtual-)machine, but as an example of parallel processing they are misleading. I would have added other examples, like pino logger, not only message queues.

• Reply • Share ›

Aram Manukyan • 8 months ago

I don'd understand the answer about "checkApiKey" question

• Reply • Share ›

Sadaharu Aram Manukyan • 7 months ago

I'll try to explain from my understanding.

When comparing strings, Node.js will compare one character by one, and terminate the comparison if there is a mismatch, which makes sense in term of optimization.

To help you understand the security concern, I'll give you an example.

Let's say your API key from DB is '123456', and the time Node.js execute a character comparison is 1 nano second.

It means checkApiKey(apiKeyFromDb, '123ABC') takes about 4 nano seconds (string compare stops when comparing '4' from first argument and 'A' from second argument)

It means checkApiKey(apiKeyFromDb, '1234CD') takes about 5 nano seconds (string compare stops when comparing '5' from first argument and 'C' from second argument)

By measuring the execution time, the attacker

Dinesh jain • a year ago

Refer http://array151.com/blog/no... for some new questions

Reply • Share ›

Md. Zahirul Haque • a year ago

Thanks for helpful questions !!

Reply • Share ›

Pablo Ruan • 2 years ago

Great content! Thanks :D

Reply • Share ›

Raf • 2 years ago

promises answer doesn't say anything about promises other that they "make your life easier" and are "better", but why? what do they accomplish?. most people are confused about promises and think that they are invented to solve callback hell or look neater. in fact promises follow a spec that makes them easy to chain (because each operation returns a promise), and removes inversion of control we have with callbacks: if you give someone a callback, you have no control how many times it is called, or if success and error get called simultaneously etc.. correctly implemented promise library removes this loophole by ensuring that it can be resolved only once and you can get either success or failure. if the spec is properly implemented each operation in promise chain gets an immutable copy of original promise. i recommend looking at the spec https://promisesaplus.com/

Reply • Share ›

visualjeff • 2 years ago

Promised can be composed. Unlike callbacks.

Reply • Share ›

Markus Westerholz • 2 years ago

Great list, thanks.

I would definitely additionally ask about

Reply • Share ›

Hire RisingStack

Outsource Development

24/7 Node.js Support

Training (fullstack & devops)

Todd Goodwin ➦ Markus Westerholz • 2 years ago

+1 on the streams. Plus, I'd add a question or 2 regarding other parts of Node's API like fs, path, process, and http. Seems like the above questions were more general programming questions and not necessarily about Node.

Reply • Share ›

Narayanan Ts • 2 years ago

Great Article! Thanks!

Reply • Share ›

Matt Gordon • 2 years ago

Hey Gergely,
Great article! I am curious, I can't seem to find the info on yarn vs npm?

Reply • Share ›

Gergely Németh ➦ Matt Gordon • 2 years ago

Thanks Matt! Actually, we were thinking about including it, but we left it out eventually. For now I would suggest keep using npm for open source libraries, so if a new update gets pushed, you will get that automatically as well, while for production application deployments yarn seems like a good fit.

1 • Reply • Share ›

Pharmokan • 2 years ago

thanks good job

Reply • Share ›

Marcelo Alves • 2 years ago

The article is great, the only problem are the examples without semicolons. Javascript have automatic semicolon insertion, but this doesn't mean you shouldn't use them. It's a bad practice, doesn't matter how much they say "it's ok", the lack of semicolons can lead to bugs that are hard to track. Just... STOP.

Reply • Share ›

semicolons, and I have to say: it's a matter of preference, as Gergely said. There's a plethora of other mistakes that could make your code fail, missing semicolons are hardly one. Especially if you have a finely configured linter that could help you with most of the pitfalls.
So, as a suggestion, *you* should stop telling other not to use them. There's no universal CS in JavaScript (not even Crockford is the unique authority), so let the others use their own.

4 • Reply • Share ›

Marcelo Alves MaxArt • 2 years ago

Sorry, won't waste any more of my time here :)

• Reply • Share ›

Gergely Németh Marcelo Alves • 2 years ago

Semicolons are just a matter of preference - you can use them or not, your call. The only gotcha with omitting semicolons is to never start lines with (, [, or `, but that's automatically checked for you when using standardjs

• Reply • Share ›

Marcelo Alves Gergely Németh • 2 years ago

Like I said, semicolon are indeed optional, but that doesn't mean you should
not use them. There are a lot of examples of errors that happen because of the omission of semicolons, just google it. Also, big names like MDN and Crockford recommend the use of semicolons. It's so