← Back

Top 125+ NodeJS Interview Questions and Answers to Crack Your Next Job

October 8, 2025

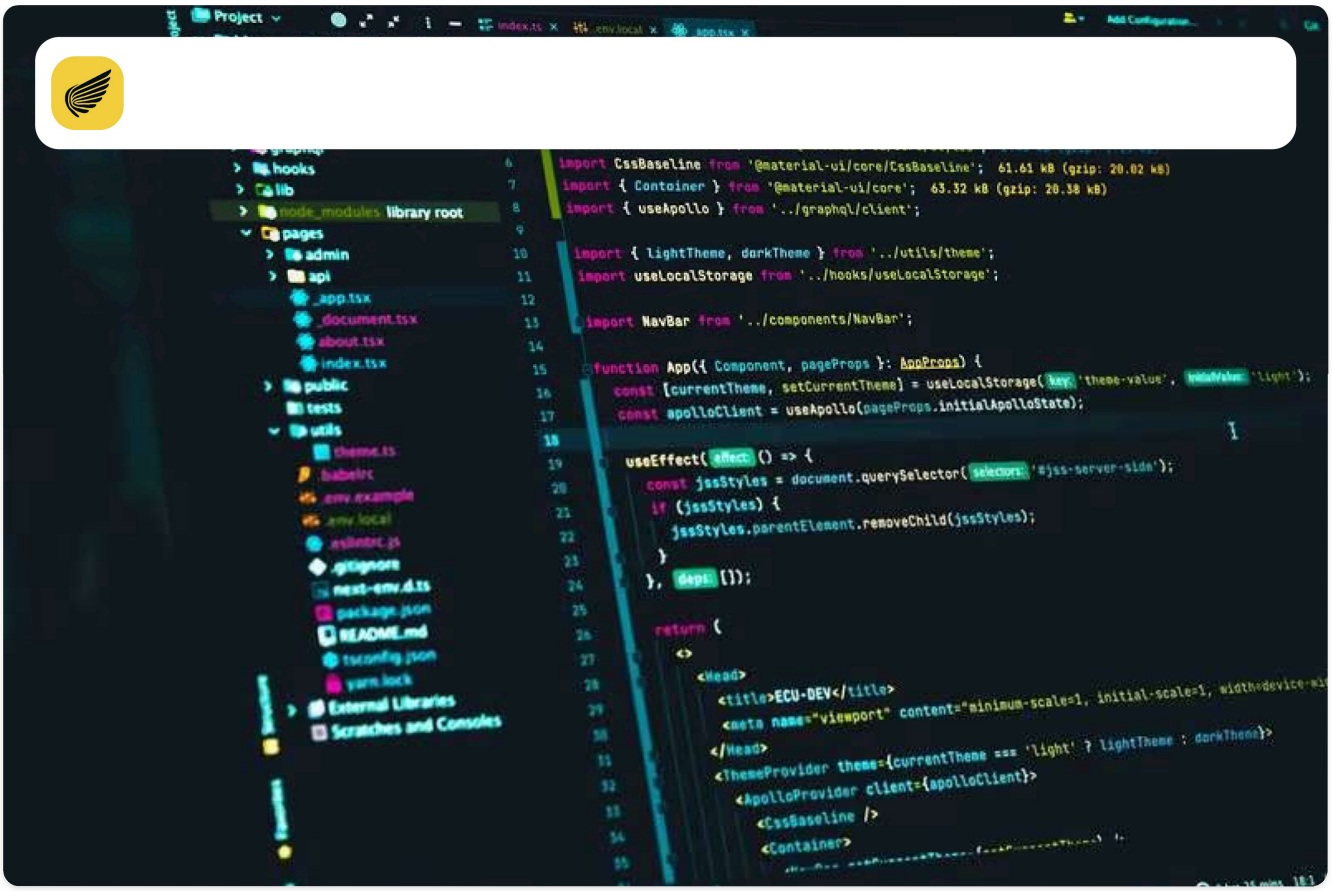
I'll never forget my first backend coding interview. I walked in thinking my LeetCode grind had me covered: binary search trees, dynamic programming, you name it. But then the interviewer asked me about the Node.js event loop, and my brain just... froze. Async patterns? Streams? Middleware chaining? Total blackout. That one moment cost me the callback.

That's when I realized: for backend and full-stack roles, your Node.js fundamentals can make or break the interview. You can crush algorithms, but if you stumble on promises, async/await, REST API design, or even how to debug a memory leak, the interviewer starts to doubt whether you can handle real-world systems.

This guide is here to save you from that. I'll break down the most common (and toughest) Node.js interview questions with clear explanations, sample answers, and insights you can actually use. We'll cover callbacks, module systems, performance tuning, testing, deployment, and more, basically everything I wish I had mastered before walking into that room.

And you don't have to prep solo. InterviewCoder's AI Interview Assistant works quietly during live interviews, feeding you real-time answers, code suggestions, and explanations while staying invisible on screen shares. No guesswork, no freezes, just steady performance under pressure.

Top 35+ NodeJS Interview Questions for Freshers



When I started interviewing for backend and full-stack roles, I assumed crushing LeetCode would carry me through. It didn't. I still remember blanking out when a Meta interviewer asked me to explain the Node.js event loop and why streams matter for performance. That moment sent me down the rabbit hole of studying the *actual* questions interviewers care about.

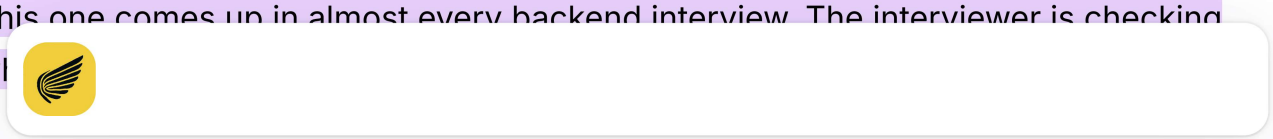
This list isn't theoretical. These are the Node.js questions that came up in interviews at Amazon, TikTok, Meta, and a dozen other companies. I've broken each one down so you know:

- What they're really testing
- How to answer with confidence
- What strong candidates say (and what weak answers miss)

Let's get into it.

1. What is Node.js, and how does it work?

This one comes up in almost every backend interview. The interviewer is checking wh



What they want to hear:

- Node.js is a server-side JavaScript runtime built on Chrome's V8 engine
- It provides access to low-level APIs like file systems, networking, and processes
- It uses an event-driven, non-blocking I/O model
- The event loop handles async tasks; blocking operations get sent to the libuv thread pool
- This model enables scalability without multi-threading

What makes an answer strong: Mention V8, event loop, libuv, and tie them to real use cases like streaming or real-time chat.

2. What tools help enforce a consistent code style?

This checks if you know how teams keep code clean and reviewable.

What to say:

- ESLint for enforcing rules and catching bugs
- Prettier for consistent formatting
- Husky + lint-staged for running checks before commits
- CI jobs and editor extensions to enforce style across environments

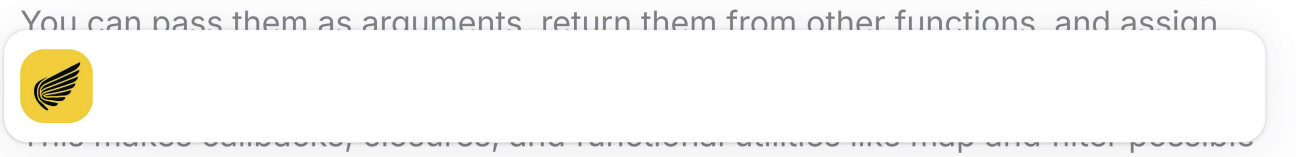
Tip: Good answers show how these tools prevent messy code in team settings. Mention Git hooks or CI pipelines if you've used them.

3. What is a first-class function in JavaScript?

If you don't understand this, you'll struggle with async logic and higher-order functions in Node.

Say this:

- First-class means functions are treated like values



A good answer includes an example:

```
const add = (a, b) ⇒ a + b;

const apply = (fn, a, b) ⇒ fn(a, b);

apply(add, 2, 3); // returns 5
```

4. Why use Node.js for backend development?

This is your chance to show you understand *where* Node fits in the backend ecosystem.

Strong answer covers:

- High performance for I/O-heavy tasks (like API calls, file ops)
- Unified language stack, JavaScript on frontend and backend
- Massive ecosystem via npm
- Great for building REST APIs, real-time apps, and CLI tools

Bonus: Acknowledge trade-offs like its poor performance with CPU-bound work and how worker threads or microservices help mitigate that.

5. How does Node.js handle requests?

This is all about showing you understand Node's internals.

What they're testing: Can you explain how the event loop interacts with the thread pool?

Strong response:

- Incoming requests are queued in the event loop

Non-blocking code runs immediately on the main thread

When those finish, their callbacks are returned to the event loop

This model supports high concurrency with minimal resources

6. Why is Node.js single-threaded?

This tests your grasp of design choices.

Say this:

Node uses a single main thread to simplify concurrency and avoid context switching

It achieves asynchronous behavior using non-blocking I/O and background threads for blocking work

The result is simpler code and efficient scaling, *as long as* you're not doing CPU-heavy tasks

7. If Node is single-threaded, how does it scale?

This is the follow-up to the last question. They want to see if you understand async patterns.

Answer like this:

The event loop handles concurrency by scheduling I/O operations without blocking the thread

libuv manages async ops behind the scenes using a thread pool

Async primitives like callbacks, promises, and async/await help coordinate tasks

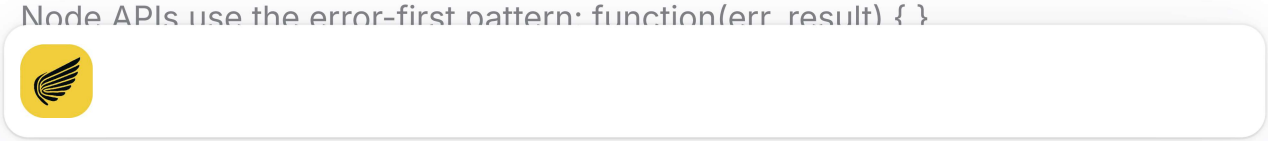
8. What is a callback in Node.js?

This one's basic, but interviewers want more than just a definition.

Say this:

A callback is a function passed to another function, to be called when an async operation finishes

Node APIs use the error-first pattern: function(err, result) { }



```
fs.readFile('file.txt', 'utf8', (err, data) ⇒ {

if (err) return console.error(err);

console.log(data);

});
```

Strong candidates also mention: callback hell and how modern code avoids it with promises or async/await.

9. Why are promises better than callbacks?

Here you're showing that you know how to write clean, maintainable async code.

Good points to hit:

- Promises flatten nested logic and make chaining easier
- .then() and .catch() allow for readable flow and centralized error handling
- They integrate with async/await for even cleaner code
- Promise utilities like Promise.all and Promise.race improve coordination

10. What does I/O mean?

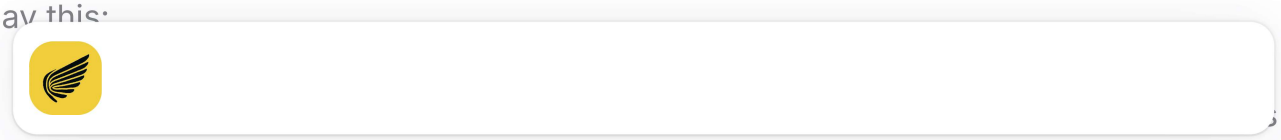
Surprisingly common question. Don't mess it up.

Answer like this:

- I/O stands for Input/Output, it is how your program interacts with external systems
- Examples: HTTP requests, reading from disk, querying a database
- Node's non-blocking model makes I/O operations lightweight and efficient

11. What is Node.js used for?

Say this:



It's also used for build tooling (like Webpack), scripting, and CLI tools

Strong fit for I/O-heavy use cases, not CPU-intensive ones

12. What's the difference between frontend and backend?

This question tests your ability to distinguish client/server roles.

Say this:

Frontend: code that runs in the browser; handles UI/UX using HTML, CSS, and JavaScript

Backend: runs on the server; handles data storage, authentication, and business logic

Frontend focuses on user interaction, backend on data and infrastructure

13. What is npm?

Solid answer:

npm = Node Package Manager

It's both a registry of packages and a CLI tool to manage dependencies

It uses package.json to track versions and scripts

package-lock.json ensures consistent installs

14. What are modules in Node.js?

This tests your knowledge of modular code.

Say this:

Modules are reusable pieces of code you import using require() or import

Node has built-in core modules like fs, http, url

You can also install third-party modules with npm

15. What does module.exports do?

Th

Say this:

- module.exports defines what gets exposed from a file when you use require()
- You can assign functions, objects, or entire classes to it

Example:

```
// math.js

module.exports.add = (a, b) ⇒ a + b;

// app.js

const math = require('./math');

math.add(2, 3); // returns 516. Why is Node.js preferred over Java or PHP?
```

This isn't about trashing other tech, it's about knowing where Node fits best.

Say this:

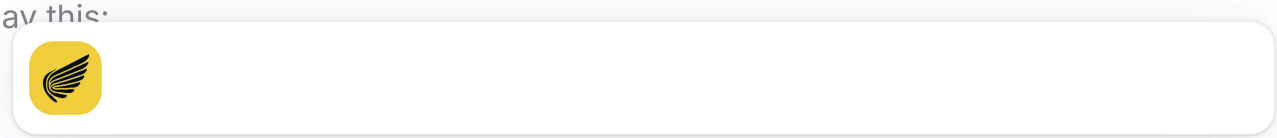
- Node is great for high-concurrency apps like APIs and real-time tools
- It allows full-stack development with JavaScript across client and server
- npm offers a massive ecosystem for rapid prototyping
- Java and PHP are better for CPU-heavy tasks or enterprise systems with complex transaction layers

A solid answer shows that you understand trade-offs, not just hype.

17. What's the difference between Angular and Node.js?

This question checks for confusion between frontend frameworks and backend runtimes.

Say this:



Applications)

Node.js is a backend runtime that runs JavaScript outside the browser

Angular runs in the browser, Node runs on the server

One handles UI and routing; the other handles APIs, file systems, and databases

18. Which database is commonly used with Node.js?

They're checking if you've worked on real projects, not just tutorials.

Say this:

MongoDB is a popular choice; its BSON format aligns well with JavaScript objects

PostgreSQL and MySQL are solid picks for relational data

Redis is commonly used for caching and session storage

Bonus: Mention Mongoose if you bring up MongoDB. It shows you understand how to model data effectively.

19. What are some commonly used Node.js libraries?

This one's about knowing the ecosystem.

Say this:

Express: minimalist web framework for handling routes and middleware

Mongoose: MongoDB ODM for defining schemas and validations

dotenv: handles environment variables

axios or node-fetch: for making HTTP requests

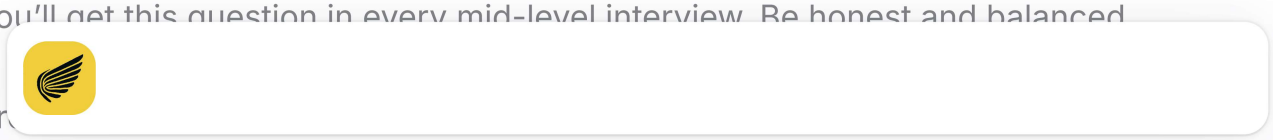
winston or pino: for logging

nodemon: for restarting the server on file changes

jest or mocha: for testing

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

You'll get this question in every mid-level interview. Be honest and balanced.



Pr

- Fast for I/O-bound workloads
- Single language across frontend and backend
- Huge ecosystem (npm)
- Non-blocking async model supports high concurrency

Cons:

- Not great for CPU-heavy computation
- Async patterns can be tricky for newcomers
- Less opinionated than some backend frameworks (e.g., Spring or Laravel)

Good answers include trade-offs and how to mitigate them.

21. What command is used to import external libraries?

They're looking for syntax accuracy.

Say this:

- In CommonJS (most Node apps): const express = require('express')
- In ES modules: import express from 'express'
- You need to run npm install express first to bring it into your project

22. Why do teams choose Node.js over Java or PHP?

Same as question 16, but interviewers love rephrasing things.

Restate clearly:

- Node is ideal for I/O-bound, real-time applications
- Unified JavaScript stack improves dev velocity
- npm helps build prototypes and products quickly

Java/PHP may be better for CPU-bound tasks or strict enterprise environments

23. [illegible] functions?

This is fundamental to writing scalable Node apps.

Say this:

- Synchronous functions block the main thread until they finish
- Asynchronous functions return control immediately and complete later
- Sync code uses try/catch for errors; async uses callbacks, promises, or async/await

Example:

```
// Synchronous

const data = fs.readFileSync('file.txt', 'utf8');

// Asynchronous

fs.readFile('file.txt', 'utf8', (err, data) ⇒ { });
```

24. What’s the purpose of require in Node.js?

Basic syntax check.

Say this:

- require() loads built-in, local, or npm modules in CommonJS
- It returns the module’s exported content so you can use it in your file

Example:

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

25. What is the V8 engine in Node.js?

They're testing if you understand what actually runs your code.

Say this:

- V8 is Google's open-source JavaScript engine (written in C++)
- It compiles JavaScript to native machine code
- Node uses V8 to run JavaScript outside the browser

Bonus points: Mention that V8 powers Chrome as well.

26. How do you handle environment variables in Node.js?

They want to see that you know how to separate config from code.

Say this:

- Access env vars via process.env.VAR_NAME
- Use a .env file locally and load it with the dotenv package
- Never commit your .env to version control
- Use secret managers in production

Example:

```
require('dotenv').config();

const port = process.env.PORT || 3000;
```

27. What is control flow in Node.js?

A slightly abstract question. Don't get lost in theory.

Say this:

- Control flow is how async tasks are ordered and coordinated
- Techniques include callbacks, promises, async/await
- Patterns include sequential flow, parallel execution with Promise.all, and race conditions with Promise.race

28. What is the event loop in Node.js?

Th

Say this:

> The event loop picks up tasks from the event queue and executes them
>
> It handles callbacks from async operations (I/O, timers, etc.)
>
> It's how Node achieves non-blocking behavior on a single thread

Example:

```
console.log('Start');

setTimeout(() ⇒ console.log('Inside timeout'), 0);

console.log('End');
```

Output:

Start

End

Inside timeout

29. What's the order in which async tasks are executed?

This one's tricky and shows up often in advanced rounds.

Say this:

> Synchronous code runs top to bottom
>
> Microtasks (like promises) run next
>
> Macrotasks (like setTimeout) run after that
>
> The event loop manages the order through its phases

30. What are the main disadvantages of Node.js?

Th

Say this:

CPU-heavy tasks block the event loop and degrade performance

Async code can become complex without structure

Dependency churn in the ecosystem can break builds

Weakness in handling complex relational data unless properly abstracted

Mitigation: worker threads, message queues, microservices, and use of typed layers with TypeScript.

31. What is REPL in Node.js?

Super basic, but don't skip it.

Say this:

REPL = Read Eval Print Loop

It's an interactive shell to run JS code in Node

Type node in your terminal to enter it

Useful for: testing snippets, debugging logic, exploring modules

32. How do you import a module in Node.js?

They want both CommonJS and ES Module formats.

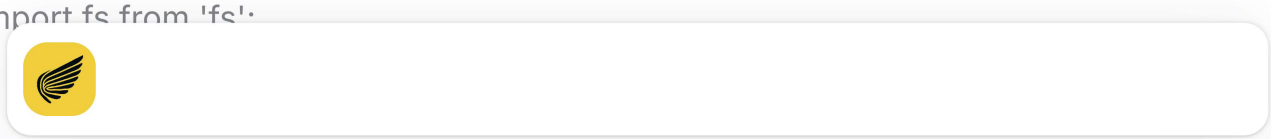
Say this:

CommonJS:

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

ES Module:

import fs from 'fs';



33. What's the difference between Node.js and Angular?

Redundant, but shows how you handle repeated questions.

Say this:

- Node.js runs JavaScript on the server
- Angular builds client-side SPAs using TypeScript
- Node handles backend logic; Angular handles UI and DOM rendering

34. What is package.json?

This is the project manifest. You should know it cold.

Say this:

- It stores project metadata, dependencies, scripts, and config
- main defines the entry point
- Scripts like "start" or "test" automate commands
- Works with package-lock.json for version locking

Example:

{

"name": "app",

"version": "1.0.0",

"main": "index.js",

"scripts": { "start": "node index.js" },

```
"dependencies": { "express": "^4.17.1" }
}
```

35. What are the most commonly used libraries in Node.js?

Just a quick name-drop test, but context helps.

Say this:

- Express: routing and middleware
- Mongoose: MongoDB ODM
- dotenv: env var config
- axios or node-fetch: HTTP requests
- winston or pino: logging
- jest or mocha: testing
- nodemon: dev-time server restarts

36. What are promises in Node.js?

A foundational async concept. You must explain clearly.

Say this:

- Promises represent values that may not be available yet
- They can be pending, fulfilled, or rejected
- Allow chaining via .then() and .catch()
- Support async/await syntax

Example:

```
async function getData() {

const res = await fetch(url);

const json = await res.json();
```

```
return json;
}
```

37. How do you import external libraries?

Yes, they'll ask the same question again with different words.

Say this:

> Install with npm install library-name
>
> Import using:

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

or

```
import express from 'express'; // ES Modules
```

> Ensure node_modules is in your project and dependencies are listed in package.json

That wraps up all 37 questions. This isn't just a quick reference, it's everything I wish I had when I was bombing early interviews. Memorize the patterns, understand the trade-offs, and most importantly: practice giving your answers out loud under time pressure.

Next step? Run through these in real mock interviews with feedback.

Download InterviewCoder today and bring it into your next interview. You'll get live AI support, from debugging help to ready-to-use code and clear explanations, so you never blank when it matters most.

Related Reading

- Vibe Coding
- Leetcode Blind 75
- C# Interview Questions

Leetcode 75

React Interview Questions

Leetcode Patterns

Java Interview Questions And Answers

Kubernetes Interview Questions

AWS Interview Questions

Angular Interview Questions

SQL Server Interview Questions

AngularJS Interview Questions

Vibe Coding

Leetcode Blind 75

C# Interview Questions

Jenkins Interview Questions

React Interview Questions

Leetcode Patterns

Java Interview Questions And Answers

Kubernetes Interview Questions

AWS Interview Questions

Angular Interview Questions

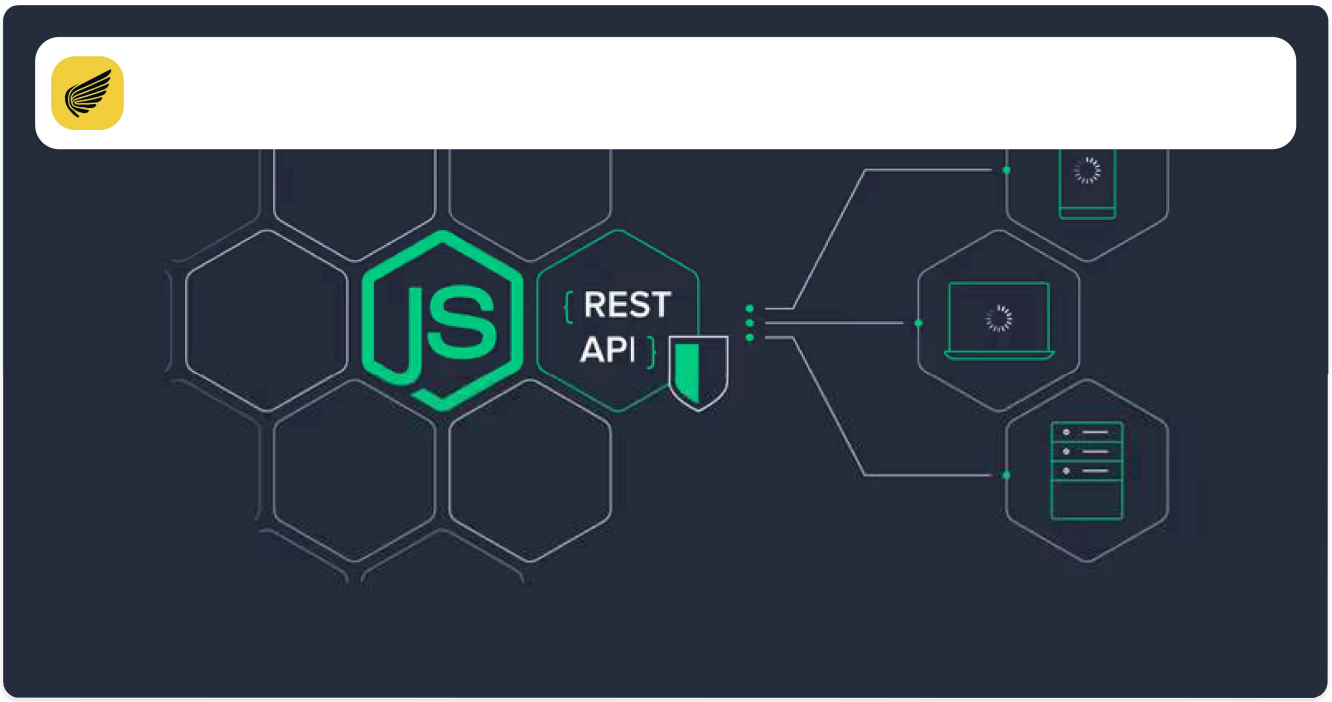
SQL Server Interview Questions

AngularJS Interview Questions

TypeScript Interview Questions

Azure Interview Questions

Top 29 NodeJS Interview Questions for Intermediate



So you've got the fundamentals down. You can talk about the event loop, async/await, and require() without flinching. But now you're interviewing for mid-level roles, and things get a bit deeper.

Interviewers stop asking if you know *what* Node.js is and start digging into *how* you use it in real-world apps. They're looking for someone who can think clearly under pressure, reason about performance, and explain trade-offs.

These 29 questions came up in my interviews at TikTok, Meta, Amazon, and in peer technical screens I've run myself. Here's how to handle them like a pro.

1. What is event-driven programming in Node.js?

This isn't just a concept to memorize. Node runs on it. Say this:

> In event-driven programming, code execution is triggered by events, not a fixed sequence
>
> You register callbacks (event handlers) for specific events, and Node's event loop fires them when ready
>
> This is what makes Node perfect for handling hundreds of I/O-bound tasks without choking

Example:

```
const EventEmitter = require('events');

co

ev.on('data', (msg) => console.log('Got:', msg));

ev.emit('data', 'hello');
```

2. What is a Buffer in Node.js?

Most devs ignore this until they deal with file streams or binary data.

Say this:

- A Buffer is a fixed-size chunk of memory used to handle raw binary data
- It's byte-oriented, unlike JS arrays or strings
- Common in file I/O, TCP sockets, or working with binary protocols

Example:

```
const buf = Buffer.from('hello', 'utf8');

console.log(buf); // <Buffer 68 65 6c 6c 6f>

console.log(buf.toString()); // 'hello'
```

3. What are Streams in Node.js?

Streams are everywhere in real backend work.

Say this:

- Streams let you process data in chunks, not all at once, which is crucial for performance
- Four types: Readable, Writable, Duplex, Transform
- Common in file transfers, HTTP responses, and real-time pipes

Example:

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

fs.
.pipe(fs.createWriteStream('out.txt'));
```

4. What is the crypto module in Node.js?

If you're not hashing passwords or signing tokens, you will be soon.

Say this:

> The crypto module handles hashing, encryption, and secure random generation
>
> Use createHash for hashing, createHmac for token signing, or randomBytes for secure tokens

Example:

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

const hash = crypto.createHash('sha256').update('hello').digest('hex');

console.log(hash);
```

5. What is callback hell?

You'll be asked this just to see if you've evolved past 2015.

Say this:

> Callback hell happens when you nest multiple async functions, making code unreadable and hard to debug
>
> The solution is to use promises or async/await

Example:

```
doThing()
```

```
.then(doNextThing)
.th
.catch(console.error);
```

6. What's the use of timers in Node.js?

Timers are essential in automation and retries.

Say this:

- setTimeout: run once after a delay
- setInterval: run repeatedly
- setImmediate: run after the I/O phase
- clearTimeout / clearInterval stop timers

Example:

```
setTimeout(() => console.log('100ms later'), 100);

setImmediate(() => console.log('on next loop'));
```

7. Difference between setImmediate() and process.nextTick()?

This is a favorite for trip-ups.

Say this:

- process.nextTick() runs before I/O, immediately after the current operation
- setImmediate() runs after I/O, on the next tick
- Overusing nextTick() can starve I/O and block the loop

Example:

```
process.nextTick(() => console.log('tick'));
```

setImmediate(() => console.log('immediate'));

8.

If you’re building APIs, you need to know this cold.

Say this:

- GET: fetch data
- POST: create new data
- PUT: replace entire resource
- PATCH: partial update
- DELETE: remove resource
- Mention idempotency: PUT is idempotent, POST is not

9. Difference between spawn() and fork()?

Important if you’re scaling with child processes.

Say this:

- spawn(): run any shell command
- fork(): spawn a new Node.js process with IPC messaging built in
- Use spawn for commands like ls, curl, etc.; fork for running Node scripts in parallel

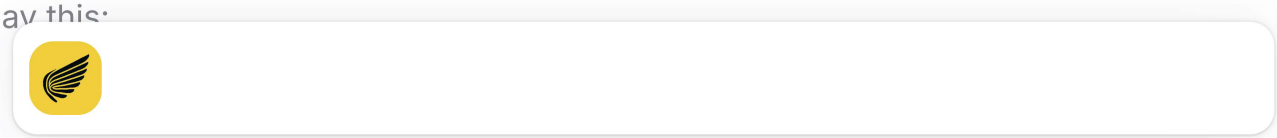
Example:

```
const { fork } = require('child_process');

const child = fork('worker.js');

child.send({ task: 'crunch' });
```

10. What’s the Passport module?

This comes up if the role involves authentication.

Say this:



Supports strategies like local, OAuth, JWT

Handles login flows, sessions, or token validation

11. What is fork() in Node.js?

A repeat of #9, but framed differently.

Say this:

fork() is used to run a new Node process and set up IPC for communication

Good for distributing CPU-heavy work or isolating risky operations

12. How to avoid callback hell?

Say this:

Promises

async/await

Generators (rare these days but still good to know)

Example:

```
async function run() {

const res = await fetchData();

const final = await process(res);

return final;

}
```

13. What is body-parser in Node.js?

Still comes up even though Express now has it built in

Sa

- body-parser was middleware to parse incoming request bodies
- Modern Express apps use express.json() and express.urlencoded() instead

14. What is CORS?

Even mid-level devs get tripped up on this.

Say this:

- CORS = Cross-Origin Resource Sharing
- It's enforced by browsers, not servers, to block frontend requests to unauthorized origins
- Servers must explicitly allow origins using headers or middleware like cors

15. What is the tls module?

If the job involves security, this might pop up.

Say this:

- tls lets you build encrypted TCP servers/clients
- It uses certificates and keys to encrypt traffic (think HTTPS without HTTP)

16. Can you access the DOM in Node?

Short answer: No.

Say this:

- Node runs outside the browser, it has no DOM
- Use jsdom if you need DOM-like APIs for testing or scraping

17. How do you manage packages in Node?

They want to see real-world workflow.

Sa

- Use npm install to manage dependencies
- Use --save-dev for dev tools
- Use package-lock.json for consistent builds
- Use npm scripts to automate tasks

18. What is NODE_ENV used for?

This controls app behavior across environments.

Say this:

- NODE_ENV is often set to development, production, or test
- You can change the config, logging, or middleware depending on it

19. What is the test pyramid?

This checks if you understand smart testing strategies.

Say this:

- Lots of unit tests, fewer integration tests, very few E2E tests
- Unit = fast and isolated
- Integration = slower, but test module interactions
- E2E = slowest, test real user flows

20. What does event-driven programming mean?

Another rephrase of #1. Be consistent and sharp.

Say this:

- You write code that responds to events, file reads, timers, and network activity
- You don't tell it when to run; the system does, based on the event loop

21. process.nextTick() vs setImmediate()

Th

Say this:

- nextTick() = before I/O
- setImmediate() = after I/O
- Abuse of nextTick() can freeze the loop

22. What are the two types of Node APIs?

They want to hear about blocking vs non-blocking.

Say this:

- Synchronous = blocks the thread
- Asynchronous = runs in the background, uses callbacks/promises
- Prefer async in servers, sync only for simple CLI or startup logic

23. What is package.json?

You've written one. Just explain it clearly.

Say this:

- Holds project metadata (name, version), dependencies, devDependencies, and scripts
- Powers npm install and npm run

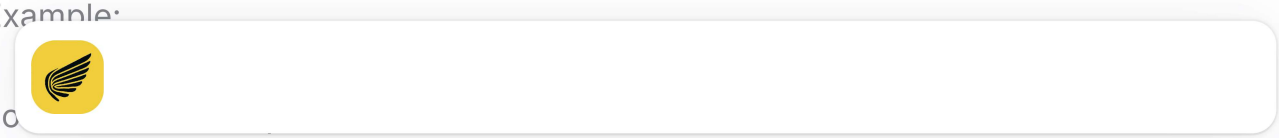
24. How do you use the URL module?

Basic but important.

Say this:

- Use the url module or URL class to parse, manipulate, and format URLs

Example:



```
const u = new URL('https://site.com/path?x=1');

console.log(u.hostname, u.searchParams.get('x'));
```

25. What is Express.js?

You've probably used it. Make sure you can explain it.

Say this:

> Minimal web framework built on top of Node's http module
>
> Handles routing, middleware, request/response parsing, static files, and more

26. How do you create a basic Express app?

Say this:

> Import Express, create an app, add middleware and routes, call listen()

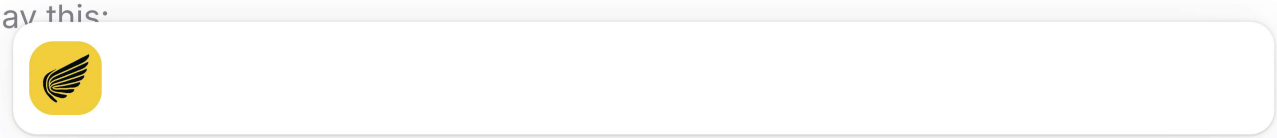
Example:

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

const app = express();

app.use(express.json());

app.get('/', (req, res) ⇒ res.send('Hello'));

app.listen(3000);
```

27. What are streams?

Repeated again, but you know this now.

Say this:



Four types: Readable, Writable, Duplex, Transform

Useful for performance in file ops, pipes, and compression

28. How do you create a simple HTTP server?

They want the raw version, no frameworks.

Example:

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

http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello World');

}).listen(3000);
```

29. What are asynchronous and non-blocking APIs?

This is Node's core value prop.

Say this:

Async, non-blocking APIs initiate I/O and return immediately

Completion is signaled with a callback, promise, or event

This model avoids blocking the single thread and lets Node scale

That wraps up the intermediate-level questions. By now, you should be able to walk into any technical screen and explain not just what Node does, but *why it works that way*, and *when to use it*.

Use InterviewCoder in your next technical screen. It gives you live coding support ins
pr

Related Reading

- Cybersecurity Interview Questions
- Leetcode Alternatives
- System Design Interview Preparation
- Ansible Interview Questions
- LockedIn
- Selenium Interview Questions And Answers
- Git Interview Questions
- jQuery Interview Questions
- Front End Developer Interview Questions
- DevOps Interview Questions And Answers
- Leetcode Roadmap
- Engineering Levels
- ML Interview Questions
- ASP.NET MVC Interview Questions
- Deep Learning Interview Questions

Top 60+ NodeJS Interview Questions for Experienced

Once you're past the basics, Node.js interviews stop being about definitions. It's about how you scale. How you handle failure. How you avoid bottlenecks, secure APIs, and write testable, production-ready code.

This list is everything I wish I had when I transitioned from junior backend dev to running real systems and interviewing at companies like Amazon, TikTok, and Meta. If you're gunning for senior roles or technical lead interviews, here's how to sound like someone who *gets it*, not just someone who Googled it.

1. What is piping in Node.js?

Piping connects readable and writable streams to process data without buffering the whole thing.

Say this:

> Use .pipe() to transfer data from a readable stream into a writable one
>
> Great for memory-efficient I/O (e.g., large files, HTTP responses)
>
> Supports chaining with transform streams and handles backpressure automatically

Example:

```
fs.createReadStream('input.log').pipe(fs.createWriteStream('output.log'))
```

2. What is a cluster in Node.js?

No

Say this:

The cluster module forks multiple worker processes that share the same server port

Each worker is a separate process with its own memory and event loop

Perfect for multi-core machines handling lots of concurrent requests

3. Cluster methods in Node.js?

Name these:

cluster.fork(): spawn a worker

cluster.isMaster: true if the current process is the master

cluster.workers: list of worker objects

worker.send() / process.on('message'): IPC messaging

Listen for 'exit', 'online', 'listening' to manage lifecycle

4. How to manage sessions in Node.js?

Sessions = stateful users. Don’t store them in memory in prod.

Say this:

Use express-session with a secure store like Redis

Store the session ID in a secure cookie

Use options like maxAge, httpOnly, sameSite, secure

5. Types of Node.js APIs?

Just two: sync and async.

Async = non-blocking (callbacks, promises, async/await)

Sync = blocking (bad for servers, fine for CLI or startup)

Never use sync in request handlers

6.

Know the difference:

- Authentication: Who are you? (login)
- Authorization: What can you do? (roles/permissions)

Use:

- passport for strategies (local, Google, etc.)
- jsonwebtoken (JWT) for stateless tokens
- Middleware for RBAC (role-based access control)

7. Tools for file uploads?

Use multer for handling multipart/form-data.

- Stores to disk, memory, or custom engines (e.g., S3)
- Support limits, filters, and custom filenames
- For stream control, busboy is a lower-level

8. Node.js vs Python?

Context is everything.

- Node: non-blocking, event-driven, great for I/O-heavy APIs
- Python: sync-first, better for CPU-heavy workloads (data science, ML)
- Node scales with cluster/worker threads, Python via multiprocessing

9. Connecting to MongoDB?

Use Mongoose or the native driver.

```
mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true })
```

Handle disconnect, retry, SIGINT for graceful shutdown

10. Reading CLI args?

- Use process.argv.slice(2)
- For flags, use minimist, yargs, or commander

```
const argv = require('minimist')(process.argv.slice(2))
```

11. Redis with Node.js?

Common use cases:

- Caching DB queries
- Rate limiting
- Session storage (with connect-redis)
- Pub/Sub for events

Use ioredis or node-redis.

12. What is WebSocket?

It's a persistent full-duplex connection between client and server.

- Use ws or socket.io in Node.js
- Ideal for chat, games, live dashboards
- Better than polling, lower latency, fewer HTTP overheads

13. What is the util module?

- promisify(): convert callbacks to promises
- inspect(): readable object printing (especially for logging)
- inherits(): old-school inheritance

14. DNS module usage?

dns.lookup: uses OS (respects /etc/hosts)

Use when you need raw DNS records

15. setImmediate() vs setTimeout()?

- setImmediate() runs after I/O
- setTimeout(fn, 0) runs after the timer phase
- Order can vary, don't rely on it deterministically

16. What is EventEmitter?

The heart of Node's async model.

```
const { EventEmitter } = require('events')

const e = new EventEmitter()

e.on('data', console.log)

e.emit('data', 'hi')
```

- Use .on(), .once(), .emit()
- Watch for memory leaks, manage listeners!

17. fork() vs spawn()?

- fork(): run another Node.js script with IPC channel
- spawn(): run any shell command (ls, curl, etc.)
- fork() = structured parallel Node apps

18. What is Buffer?

- Stores raw binary data (e.g., from files or sockets)
- Not resizable
- Use Buffer.alloc() or Buffer.from()

19. What is piping?

Se

20. Why V8?

- Compiles JS to machine code (JIT)
- Fast, optimized, embeddable
- Garbage-collected, battle-tested in Chrome

21. What is middleware?

Functions like this:

(req, res, next) ⇒ { ...; next() }

- Used in Express for logging, auth, parsing, etc.
- Execution is ordered, so placement matters

22. HTTP Methods?

- GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
- Understand idempotency, safe methods, and status codes

23. Connect MongoDB to Node.js?

See #9. Use the native driver or Mongoose.

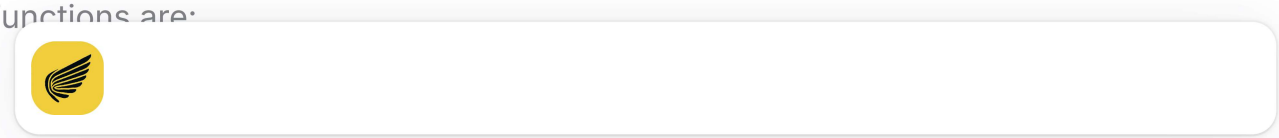
24. What is WASI?

WebAssembly System Interface

- Let WASM run securely in Node with access to files, env, etc.
- Example: sandboxed plugin systems

25. First-class functions?

Functions are:



Passed as arguments

Returned from functions

Use case: callbacks, higher-order functions, middleware

26. How to manage packages?

Use npm or yarn

Scripts (test, start, lint)

Lockfiles (package-lock.json) = deterministic installs

Use npm audit for security

27. Why use Node?

One language front to back

Non-blocking I/O = great for APIs

Huge ecosystem

Fast prototyping

28. async.queue arguments?

First: worker function (task, cb)

Second: concurrency number

29. Purpose of module.exports?

Expose things from a module:

module.exports = { add }

Don’t confuse with exports = ... (this breaks the link).

30. Async vs sync functions?

Sync blocks the event loop

Always prefer async for server code

31. Async tasks in event loop?

I/O: file, DB, HTTP

Timers: setTimeout

Microtasks: Promises, queueMicrotask

Immediate: setImmediate

32. Execution order in control flow?

Sync code runs top-down

Promises (microtasks) before timers

nextTick runs before everything

33. Inputs to async.queue?

Task objects (e.g., { id: 1 })

The worker function processes each task

Use .push() or .unshift()

34. Node.js disadvantages?

Not ideal for CPU-heavy tasks

Single thread = blocked by sync code

Needs clustering or worker threads to scale compute

35. Node.js vs AJAX?

Node.js = server-side runtime

AJAX = browser-side technique for async HTTP calls

They work together: frontend AJAX calls Node.js APIs

36. What is "non-blocking"?

Node excels at non-blocking I/O: files, sockets, DBs.

37. How does Node avoid blocking?

- Uses libuv for async I/O
- Delegates blocking work to the thread pool
- Event loop coordinates callbacks

38. How to use async/await?

```
async function main() {

try {

const data = await fetchData()

} catch (err) {

console.error(err)

}

}
```

- Await only inside async functions
- Handle errors with try/catch

39. Why separate the Express app and server?

- Easier testing with supertest
- Cleaner deployment (serverless, multiple protocols)
- Better code organization

40. Node.js security features?

Use Helmet for secure headers

Validate inputs

Avoid eval

Use dotenv or vaults for secrets

41. What is the test pyramid?

Unit tests (most): test individual logic

Integration tests: test component interaction

E2E tests: test full user flow (slowest)

Tools:

jest, supertest, cypress, playwright

42. What is EventEmitter?

See #16. Add:

Max listener warnings

Use .removeListener() to prevent leaks

43. How to scale with a cluster?

Fork workers with cluster.fork()

Restart crashed workers

Use load balancing and sticky sessions if needed

44. What's the thread pool?

4 threads by default (can increase with UV_THREADPOOL_SIZE)

Used for fs, DNS, crypto ops

Managed by libuv

45. Worker threads vs clusters?

Clusters: separate processes, isolated memory

Threads = better for compute tasks; clusters = better for scaling I/O

46. Measure async durations?

```
const { performance } = require('perf_hooks')

const start = performance.now()

// await something

console.log(performance.now() - start)
```

47. Measure performance?

Use --inspect, --prof, clinic.js, or benchmark.js

For async tracing: OpenTelemetry

48. Stream types?

Readable

Writable

Duplex (both)

Transform (modifies data, e.g., compression)

49. What is tracing?

Tracks performance and flow across services.

Tools: OpenTelemetry, New Relic

Useful for diagnosing latency bottlenecks in async code

50. What is package.json?

Project manifest

Powers npm install, npm run, and npm publish

51. readFile vs createReadStream?

readFile: loads entire file into memory

createReadStream: streams in chunks (better for big files)

52. Uses of crypto?

Hashing: createHash

HMAC

AES encryption

Secure tokens: crypto.randomBytes

53. What is passport?

Auth middleware for Express

Supports local, JWT, and OAuth strategies

Session or token-based auth flows

54. Get file info?

```
const stats = await fs.promises.stat('file.txt')

console.log(stats.size, stats.mtime)
```

55. DNS lookup?

dns.lookup: OS resolver

dns.resolve: DNS-level query

Use based on caching vs fresh records

56. setImmediate() vs setTimeout()?

See #15, #21, #56. Be consistent: phases matter, behavior varies.

57.

Converts Unicode domain names to ASCII

'münich.com' → 'xn--mnich-kva.com'

Important for DNS compatibility

58. Debugging in Node.js?

node inspect, --inspect-brk

Attach Chrome DevTools or VS Code

Use heap snapshots and profilers

59. Is crypto supported?

Yes, via the crypto module.

Use secure algorithms

Never roll your own

Rely on platform-tested features

60. Why are you the right fit for this Node.js role?

Say something like:

I've built production-scale Node.js APIs with clustering, Redis caching, MongoDB, JWT auth, and CI pipelines. I focus on performance, test coverage, and DX. I've led code reviews, deployed with Docker, and handled on-call issues. I know how to write code that ships, scales, and handles failures gracefully.

61. Do you have Node.js experience?

List your wins clearly:

REST APIs with Express + Mongo

Auth flows with Passport + JWT

Optimized file streaming with backpressure

Logged + monitored using Winston, ELK, or Datadog

Nail Coding Interviews with our AI Interview Assistant: Get Your Dream Job Today

Grinding LeetCode for months only to stumble in one tech screen? That playbook is outdated. InterviewCoder is your invisible AI assistant, running silently in the background during interviews. It's completely undetectable and gives you:

- Live code suggestions
- Debugging guidance
- Complexity explanations

All while you focus on clear communication with your interviewer.

Over 87,000 developers have already used InterviewCoder to land offers at FAANG, Big Tech, and leading startups. This is how smart engineers skip the burnout and walk into interviews with confidence, not anxiety.

Download InterviewCoder today. Show up sharp. Leave with the offer.

← Back to Blog

Interview Coder

Interview Coder is a desktop app designed to help job seekers ace technical interviews by providing real-time assistance with coding questions.

All systems online

Legal

Refund Policy

Terms of Service

Cancellation Policy

Pages

Contact Support

Create account

Login to account

Affiliate Program

Leetcode Problems

Compare Offers

Software Engineer Salaries

Software Engineer Resume

Compare

Final Round AI

AIApply
Alternative

UltraCode
Alternative

LockedIn AI
Alternative

Interviewing.io
Alternative

Formation.Dev
Alternative