UnCodemy

# Node.js Interview Questions PDF

Node.js interviews test the event loop and non-blocking I/O, modules, callbacks, promises and async await, streams, Express middleware, error handling, environment configuration, security basics and testing. This page gives short model answers with small examples so you can practise explaining how Node handles many requests on one thread.

## Key takeaways

- Node runs JavaScript on a single main thread and offloads I/O so it can serve many requests.
- Blocking the event loop with heavy computation slows every request.
- Express middleware runs in order and shapes every request and response.
- Handle errors in async code explicitly and never leak internal details to clients.
- Validate input, keep secrets in environment variables and update dependencies.

## What is Node.js interview questions?

Node.js is a JavaScript runtime built on the V8 engine that lets developers run JavaScript on servers, using an event-driven, non-blocking model suited to handling many concurrent network requests.

## What is Node.js and when is it a good fit?

Node.js runs JavaScript outside the browser on the V8 engine. It is a good fit for I/O-heavy services such as APIs, real-time apps and tools that wait on networks and databases. It is less suitable for long CPU-heavy work on the main thread, which should go to worker threads or separate services.

## How does the event loop work?

Node runs your JavaScript on one main thread. When it starts an operation such as reading a file or a network call, it hands the work to the system or a thread pool and continues. When the work finishes, the callback is queued and the event loop runs it when the call stack is empty. The loop has phases (timers, pending callbacks, poll, check and close). Promise callbacks (microtasks) and `process.nextTick` run between phases, before the next timer.

```
console.log("A");
setTimeout(() => console.log("B"), 0);
Promise.resolve().then(() => console.log("C"));
console.log("D");
// A, D, C, B
```

## What does non-blocking mean?

A non-blocking call returns immediately and signals completion later, so the thread can serve other requests. A blocking call such as `fs.readFileSync` or a long loop stops everything until it finishes. Use async versions in servers.

## How do modules work?

CommonJS uses `require` and `module.exports`. ES modules use `import` and `export` and are enabled with `"type": "module"` in `package.json` or the `.mjs` extension. Modules are cached after the first load, so a module's top-level code runs once.

## Callbacks, promises and async await?

Callbacks are functions passed to be called later; nesting them gets hard to read. Promises represent a future value and chain with `.then`. `async` and `await` give a sequential style on top of promises, with `try/catch` for errors. Run independent tasks together with `Promise.all`.

## What are streams?

Streams process data in chunks instead of loading it all in memory, for example reading a large file or piping a file to an HTTP response. Types are readable, writable, duplex and transform. Streams reduce memory use and start sending data sooner.

## What is Express middleware?

Middleware functions receive the request, the response and a `next` function. They run in the order registered and can modify the request, end the response or call `next`. Common uses are logging, parsing JSON, authentication and error handling.

```
app.use(express.json());
app.get("/users/:id", async (req, res, next) => {
  try {
    const user = await getUser(req.params.id);
    if (!user) return res.status(404).json({ error: "Not found" });
    res.json(user);
  } catch (err) { next(err); }
});
app.use((err, req, res, next) => res.status(500).json({ error: "Server error" }));
```

## How should I handle errors?

Use `try/catch` with async code, pass errors to a central error handler, return correct status codes and log details on the server only. Handle `unhandledRejection` and `uncaughtException` by logging and shutting down cleanly, because the process may be in an unsafe state.

## How do I configure an app?

Read settings from environment variables, keep secrets out of source control, provide an example file without real values and validate required settings at startup. Use different settings for development and production.

## What security basics should I know?

Validate and sanitise input, use parameterised database queries to prevent injection, hash passwords with a strong algorithm, set secure HTTP headers, limit request rates, keep dependencies updated and avoid exposing stack traces. The OWASP Top Ten is a good checklist.

## How do I scale a Node app?

Run several processes with a process manager or the cluster module to use all CPU cores, put a load balancer in front, keep the app stateless by moving sessions to a shared store and use caching for repeated reads.

## How do I test a Node service?

Write unit tests for functions and integration tests that call the API with a test client against a test database. Mock external services, and run the tests in CI.

## What small coding tasks are common?

Typical tasks are writing a REST endpoint that returns JSON, reading a file with streams, adding authentication middleware, implementing a rate limiter or reading from a database and handling failure. Explain the routes, the status codes you return, how you validate input and where you handle errors. Mention how you would test the endpoint with a client tool.

## How do I discuss performance?

Measure before changing: use logging and profiling to find slow queries or blocking code. Typical fixes are adding database indexes, caching repeated reads, using streams for large payloads, compressing responses, moving heavy computation to workers and using connection pools. Always state the trade-offs, such as cache staleness.

## Frequently asked questions

### What is Node.js?

Node.js is a JavaScript runtime based on the V8 engine that runs JavaScript on servers using an event-driven, non-blocking model.

### How does the Node.js event loop work?

The main thread runs your code and hands I/O to the system. When work finishes, callbacks are queued and run when the call stack is empty.

### What is middleware in Express?

Middleware functions receive the request, response and next, run in order and handle tasks such as parsing, logging and authentication.

### Why avoid synchronous functions in a server?

They block the single main thread, so every other request waits until the call finishes.

### How can Node use multiple CPU cores?

Run several processes with a process manager or the cluster module, or use worker threads for CPU-heavy tasks.

## Sources and further reading

- Node.js documentation
- Express documentation
- OWASP Top Ten

## About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550