Node.js Engineer Study Guide

Express.js & API Development

Comprehensive Study Guide with Detailed Explanations, Examples, and Diagrams

1. Core Node.js Internals

Q: What is Node.js?

Node.js is a JavaScript runtime built on Chrome's V8 engine for server-side applications.

Q: Event-driven & Non-blocking I/O

Node.js uses an event-driven architecture and non-blocking I/O to handle multiple operations concurrently.

Q: Event Loop Phases

Event loop executes callbacks in different phases: timers, pending callbacks, idle, poll, check, close callbacks.

Q: Example: Async File Read

const fs = require('fs'); fs.readFile('data.txt', 'utf8', (err, data) => { if(err) throw err; console.log(data); }); console.log('Reading file...');

2. Express.js & API Development

Q: What is Express.js?

Express.js is a minimal and flexible Node.js web framework for building APIs and web apps.

Q: Middleware

Middleware functions process requests and responses in the order they are added. Can be custom or built-in.

Q: Example Middleware

app.use((req, res, next) => { console.log(`${req.method} ${req.url}`); next(); });

Q: Error Handling

app.use((err, req, res, next) => { console.error(err); res.status(500).json({error:'Something went wrong'}); });

Q: JWT Auth Example

const jwt = require('jsonwebtoken'); app.post('/login', (req,res) => { const user={id:1,username:'john'}; const token = jwt.sign(user,'secretKey',{expiresIn:'1h'}); res.json({token}); });

3. Advanced Node.js Topics

Q: Streams & Backpressure

Streams handle large data efficiently. Backpressure occurs when the consumer cannot process data as fast as it's produced.

Q: Worker Threads

Worker threads allow parallel execution for CPU-intensive tasks.

Q: Clustering

Cluster module allows Node.js to run multiple instances to utilize multi-core CPUs.

Q: Caching

Redis can be used for caching responses or frequently accessed data.

Q: Memory Management

V8 engine handles garbage collection. Avoid memory leaks by cleaning unused references.

4. Security & Deployment

• Use Helmet.js for secure HTTP headers.

• Implement rate-limiting to prevent brute-force attacks.

• Validate and sanitize all user inputs.

• Use HTTPS for secure communication.

• Keep dependencies updated using npm audit.

• Dockerize Node.js apps and deploy with PM2 or Nginx.

5. Testing & Monitoring

• Use Jest for unit testing and Supertest for API testing.

• Log application data with Winston or Morgan.

• Monitor processes using PM2 and aggregated logs for production.

6. Quick Revision Sheet

• Event loop phases and async behavior

• Express.js middleware and error handling flow

• JWT authentication and refresh tokens

• Streams, backpressure, and caching strategies

• Clustering and PM2 deployment tips

• Security best practices: Helmet, rate-limiting, input validation