Node.js & Express.js Revision Questions

Beginner Level

Q1: What is Node.js?

Answer:
Node.js is a runtime environment that allows JavaScript to run outside the browser. It is built on Chrome's V8 engine and is used for building scalable server-side applications.

Q2: What is Express.js?

Answer:
Express.js is a minimal and flexible web framework built on top of Node.js. It simplifies building web applications and APIs by providing routing, middleware support, and HTTP utilities.

Q3: What is the Event Loop in Node.js?

Answer:
The Event Loop allows Node.js to perform non-blocking I/O operations despite being single-threaded. It processes callbacks from the queue once the call stack is empty.

Q4: What are modules in Node.js?

Answer:
Modules are reusable blocks of code. Node.js has:

- Core modules (fs, http, path)
- Local modules

- Third-party modules (installed via npm)

Example:

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

Q5: What is middleware in Express?

Answer:
Middleware functions are functions that have access to:

- request (req)
- response (res)
- next()

They execute during the request-response cycle.

Example:

```
app.use((req, res, next) => {
  console.log("Middleware executed");
  next();
});
```

Intermediate Level

Q6: What is the difference between `require()` and `import`?

Answer:

- `require()` → CommonJS (default in Node.js)

- `import` → ES Modules (modern syntax)

Q7: What is the difference between synchronous and asynchronous methods in Node.js?

Answer:

- Synchronous → Blocks execution
- Asynchronous → Non-blocking, uses callbacks/promises

Example:

```
fs.readFileSync(); // blocking
fs.readFile();     // non-blocking
```

Q8: What is routing in Express?

Answer:
Routing refers to defining endpoints that respond to client requests.

Example:

```
app.get('/users', (req, res) => {
  res.send("User list");
});
```

Q9: What is `next()` in Express?

Answer:
`next()` passes control to the next middleware function in the stack. If not called, the request will hang.

Q10: What is the difference between `res.send()`, `res.json()`, and `res.end()`?

Answer:

- `res.send()` → Sends response (auto-detects type)
- `res.json()` → Sends JSON response
- `res.end()` → Ends response without body

Advanced Level

Q11: How does Node.js handle concurrency if it is single-threaded?

Answer:
Node.js uses:

- Event Loop
- Non-blocking I/O
- libuv thread pool (for file system, DNS, etc.)

Heavy operations are delegated to background threads, and callbacks are pushed to the event queue.

Q12: What is error-handling middleware in Express?

Answer:
Error middleware has 4 parameters:

```
app.use((err, req, res, next) => {
  res.status(500).send(err.message);
});
```

It handles errors globally.

Q13: What is the difference between `process.nextTick()` and `setImmediate()`?

Answer:

- `process.nextTick()` → Executes before next event loop phase
- `setImmediate()` → Executes in the check phase

Q14: What is CORS and how do you enable it in Express?

Answer:
CORS (Cross-Origin Resource Sharing) allows cross-domain requests.

Install middleware:

```
npm install cors
```

Use:

```
const cors = require('cors');
app.use(cors());
```

Q15: What is JWT authentication?

Answer:
JWT (JSON Web Token) is a token-based authentication mechanism where the server generates a signed token and sends it to the client. The client sends it in headers for protected routes.

Expert Level

Q16: What is clustering in Node.js?

Answer:
Cluster module allows Node.js to create multiple worker processes to utilize multi-core systems.

Q17: What is rate limiting and how do you implement it?

Answer:
Rate limiting restricts repeated requests from the same client.

Example using middleware:

```
const rateLimit = require("express-rate-limit");
```

Q18: What is the difference between blocking code and CPU-intensive code?

✅ Answer:

Although both can affect performance, they are not the same.

🔹 1️⃣ Blocking Code

Blocking code is code that stops the execution of other operations until it finishes.
It blocks the event loop, preventing Node.js from handling other requests.

📌 Usually caused by:

- **Synchronous I/O operations**
- **Long-running synchronous functions**

Example (Blocking):

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


const data = fs.readFileSync('file.txt'); // blocks event
loop

console.log(data.toString());
```

Here, the server cannot process other requests until the file is fully read.

🔹 2️⃣ **CPU-Intensive Code**

CPU-intensive code performs heavy computations that consume high CPU resources.
Even if written asynchronously, heavy computation can still block the single thread.

📌 **Examples:**

- **Image processing**
- **Encryption**
- **Large loops**
- **Data analysis**

Example (CPU-Intensive):

```
for (let i = 0; i < 1e9; i++) {
```

```
  // heavy calculation
}
```

This blocks the event loop because JavaScript runs on a single thread.

🔥 Key Difference

Blocking Code	CPU-Intensive Code
Usually caused by synchronous I/O	Caused by heavy computations
Can be avoided using async APIs	Requires worker threads or clustering
Related to I/O	Related to CPU usage

✅ How to Handle CPU-Intensive Tasks?

- Use `worker_threads`
- Use `child_process`
- Use clustering
- Offload to microservices

⭐⭐⭐⭐ Q19: Explain Streams and Piping in Node.js.

What are Streams?

Streams are objects that allow you to read or write data in chunks, instead of loading everything into memory at once.

They are useful for:

- **Large files**
- **Real-time data**
- **Video streaming**
- **File uploads/downloads**

🔹 **Types of Streams in Node.js**

1. **Readable → Read data**
2. **Writable → Write data**
3. **Duplex → Read & Write**
4. **Transform → Modify data while reading/writing**

Example: Reading a File Using Stream

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

```
const readStream = fs.createReadStream('largefile.txt');


readStream.on('data', (chunk) => {
  console.log('Received chunk:', chunk.length);
});
```

Instead of loading the entire file, it reads small chunks.

What is Piping?

Piping connects the output of one stream directly to another stream.

Think of it like:

```
Readable Stream → Writable Stream
```

🔹 **Example: Piping Data**

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


const readStream = fs.createReadStream('input.txt');
const writeStream = fs.createWriteStream('output.txt');
```

```
readStream.pipe(writeStream);
```

Here:

- **Data is read from `input.txt`**
- **Directly written to `output.txt`**
- **No need to manually handle chunks**

🔥 Why Streams Are Important?

- **Memory efficient**
- **Faster performance**
- **Ideal for large data**
- **Non-blocking**

Q20: How would you structure a large-scale Express application?

Example structure:

```
controllers/
routes/
models/
middlewares/
services/
config/
```

Intermediate – Concept Strengthening

Q21: What is `package.json`?

Answer:
`package.json` is a metadata file that contains:

- Project name & version
- Dependencies
- Scripts
- Main entry file
- Author & license

It helps manage project configuration and dependencies.

Q22: What is the difference between dependencies and devDependencies?

Answer:

- **dependencies** → Required in production
- **devDependencies** → Required only in development (e.g., nodemon, jest)

Install dev dependency:

```
npm install nodemon --save-dev
```

Q23: What is `npm` and `npx`?

Answer:

- `npm` → Installs packages
- `npx` → Executes packages without globally installing them

Example:

```
npx create-react-app myapp
```

Q24: What is `process` object in Node.js?

Answer:
`process` is a global object that provides information about the current Node.js process.

Examples:

```
process.env
process.argv
process.exit()
```

Q25: What are environment variables?

Answer:
Environment variables store configuration outside the code (e.g., DB passwords, API keys).

Use:

```
process.env.PORT
```

Usually managed with `dotenv`.

Advanced Level -Concept Strengthening

Q26: What is the difference between `app.use()` and `app.get()`?

Answer:

- `app.use()` → Used for middleware (runs for all HTTP methods)
- `app.get()` → Handles only GET requests

Q27: What is Express Router?

Answer:
Express Router allows modular route handling.

Example:

```
const router = require('express').Router();

router.get('/', (req, res) => {
  res.send("Home route");
});

module.exports = router;
```

Q28: What is body-parser?

Answer:
Middleware that parses incoming request bodies.

Now built-in:

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

Q29: What is Helmet in Express?

Answer:
Helmet is a middleware that secures Express apps by setting HTTP headers.

```
const helmet = require('helmet');
app.use(helmet());
```

Q30: What is Morgan?

Answer:
Morgan is HTTP request logger middleware for Node.js.

```
const morgan = require('morgan');
app.use(morgan('dev'));
```

Expert Level -Concept Strengthening

Q31: What is the difference between `spawn`, `exec`, and `fork`?

Answer:

- `spawn()` → Launches new process (streamed output)
- `exec()` → Runs command and buffers output
- `fork()` → Special spawn for Node modules

Q32: What is worker_threads in Node.js?

Answer:
Worker threads allow running JavaScript in parallel threads for CPU-intensive tasks.

Useful for:

- Image processing
- Heavy calculations
- Data encryption

Q33: What is the difference between `readFile` and streams?

Answer:

- `readFile()` → Loads entire file into memory
- Streams → Process file in chunks (memory efficient)

Used for large files.

Q34: What is the difference between authentication and authorization?

Answer:

- Authentication → Who are you?
- Authorization → What are you allowed to do?

Q35: What is middleware chaining?

Answer:
Executing multiple middleware functions in sequence using `next()`.

Example:

```
app.use(authMiddleware);
app.use(roleMiddleware);
```

Q36: What are HTTP status codes commonly used?

Answer:

- 200 → OK
- 201 → Created
- 400 → Bad Request

- 401 → Unauthorized
- 403 → Forbidden
- 404 → Not Found
- 500 → Server Error

Q37: What is REST API?

Answer:
REST (Representational State Transfer) is an architectural style for designing APIs using HTTP methods:

- GET
- POST
- PUT
- DELETE

Q38: What is rate limiting and why is it important?

Answer:
Rate limiting restricts repeated API requests to prevent:

- DDoS attacks
- Brute force attacks
- Server overload

Q39: What is graceful shutdown in Node.js?

Answer:
Gracefully stopping server by closing open connections before exiting.

Example:

```
process.on('SIGINT', () => {
  server.close(() => {
    console.log('Server closed');
  });
});
```

Q40: How would you improve performance of a Node.js application?

Answer:

- Use clustering
- Enable compression
- Use caching (Redis)
- Use streams
- Optimize database queries
- Use load balancer
- Avoid blocking code