🟢 **Beginner Level (Core Node.js Fundamentals)**

- What is Node.js
- Core modules in Node.js
- Node architecture overview
- HTTP in detail
- parts of HTTP response
- default HTTP port (80 / 443)
- HTTP status codes (200, 201, 401, 403, 500)
- app.use() vs app.set()
- express.static()
- express.urlencoded
- Express.set
- configuring app settings
- os module
- path module (absolute vs relative path)
- environment variables
- using environment variables
- environment variables without using .env
- exporting in CommonJS
- CommonJS module system
- res.writeHead
- res.setHeader / setHeader
- writeHead vs setHeader
- app.locals vs res.locals
- Express basics (app.get, app.post)
- using body parser / express.json()
- app.all()
- OPTIONS method
- HTTP OPTION request
- delete a file using fs

- fs.existsSync()
- fs.unlink()
- fs.link()

🟡 **Intermediate Level (Routing, Middleware & Server Concepts)**

- query params vs path params
- path parameters
- dynamic routing
- reading query params and path params
- router chaining
- types of middleware
- Express middleware
- custom middleware
- express.urlencoded
- express.static() and public folder setup
- app.use() middleware chaining
- CORS
- preflight request (OPTIONS)
- configuring and using path params
- Content negotiation
- reverse proxy
- why reverse proxy was used in hosting
- JWT (basic understanding)
- JWT claims
- JWT algorithm
- how JWT signature helps in verification
- refresh token
- access token vs refresh token
- token introspection
- API versioning
- HTTP 403, 201 codes

- res.render()

🟠 Intermediate–Advanced (Asynchronous, Events & System Level Concepts)

- asynchronous programming (callbacks, promises, async/await)
- timer functions
- i/o bound vs cpu bound
- how Node handles I/O bound vs CPU bound operations
- event module (EventEmitter)
- event.on / event.emit
- create a custom event emitter
- process.nextTick
- setImmediate vs process.nextTick
- microtask vs macrotask queue
- reactor pattern
- phases in event loop
- child_process (basics)
- create child_process
- fork vs spawn
- exec vs execFile
- cluster module
- clustering (need clarity)
- socket vs socket.io
- using socket.io for real-time communication
- express.urlencoded middleware
- express.json()
- write head, set header (HTTP streaming)
- fs.readFile() vs fs.createReadStream()
- promise with fs (reading/writing files)
- create endpoint for read content from file
- create endpoint for write content into file

🔵 Advanced (Performance, Security, Concurrency & System Design)

- concurrency in Node.js
- Parallelism
- worker threads
- thread vs process
- thread pool
- cluster vs worker thread
- load balancing via cluster module
- CPU-bound vs I/O-bound task handling
- process vs thread (libuv thread pool)
- libuv internals (thread pool, event loop phases)
- event loop phases (Timers, I/O callbacks, Poll, Check, Close)
- rate limiting
- types of rate limiting (fixed window, sliding window, token bucket)
- promisify (util.promisify)
- cron-job (scheduling tasks)
- block request based on custom header
- run a function with and without setImmediate
- socket programming fundamentals
- reverse proxy setup (e.g., Nginx + Node.js)
- DNS fundamentals (lookup, caching)
- process environment variables (process.env)
- API security (rate limiting, helmet, CORS, CSRF)
- CSRF (how malicious sites implement CSRF attacks)
- XSS attack
- how to secure Node.js APIs
- refresh token strategy
- token introspection
- secure cookies (httpOnly, SameSite, Secure flags)
- JWT.verify() example
- res.locals usage for passing data between middleware

● **Expert (Low-Level Internals, Scaling & Optimization)**

- libuv deep dive (event loop internals)
- Node.js reactor pattern in depth
- thread pool tuning (UV_THREADPOOL_SIZE)
- worker threads (heavy computation handling)
- child_process internals (fork/spawn/exec/execFile differences)
- inter-process communication (IPC)
- Node.js clustering in production (round-robin, master-worker setup)
- scaling Node apps (horizontal & vertical)
- reverse proxy (Nginx/PM2/Load Balancer setup)
- Node concurrency model (single-threaded + libuv multi-threaded pool)
- caching (Redis, in-memory strategies)
- streaming (Readable/Writable/Transform/Duplex)
- transform stream vs duplex stream
- backpressure handling
- Node performance optimization (async I/O, worker threads)
- memory leaks detection & prevention
- monitoring with Node Profiler
- security best practices (rate limiting, helmet, JWT expiry, CSRF)
- debugging (inspector, Chrome DevTools)
- process management (PM2, forever)
- logging & tracing (Winston, Morgan)
- environment-based configurations (dotenv, process.env)
- Node.js with reverse proxies (Nginx setup)
- DNS, TCP sockets, HTTPS module
- advanced token handling (JWT introspection, refresh flow)