Backend Interview Questions Answers

Node.js (Q1–Q22)

Q1. Node.js kya hai aur kyun use hota hai?

Ans: Node.js ek **JavaScript runtime** hai jo V8 engine par chalti hai. Ye server-side JS run karne deta hai. Reasons: non-blocking I/O, high concurrency handle karta hai, single language (front + back), rich ecosystem (npm). Real-time apps (chat, notifications), APIs, streaming services me best.

Q2. Event loop kya hota hai Node.js me?

Ans: Event loop Node ka **orchestrator** hai jo asynchronous tasks (timers, I/O, promises, microtasks) ko phases me process karta hai. Ye hi Node ko **non-blocking** banata hai. Heavy CPU tasks event loop ko block kar dete hain, isliye ya to worker threads/child processes use karo ya microservices.

Q3. Single-threaded hone ke bawajood Node.js concurrent kaise hai?

Ans: JS execution single thread pe hoti hai, lekin I/O operations **libuv** thread pool aur OS ke saath offload hote hain. Event loop callbacks ko schedule karke concurrency achieve karta hai.

Q4. CommonJS vs ES Modules difference?

Ans:

- CommonJS: `require`, `module.exports`, synchronous, default in older Node.
- ESM: `import`, `export`, static analysis friendly, tree-shaking.

Modern projects ESM prefer karte hain (`"type": "module"`).

Q5. Callback hell kya hai aur kaise avoid karte ho?

Ans: Nested callbacks jo unreadable ho jate hain. Avoid via **Promises**, **async/await**, aur control flow libraries. Error handling try/catch + centralized handlers se.

Q6. Stream kya hota hai Node me?

Ans: Streams **chunked data** handle karte hain (readable, writable, duplex, transform). Memory-efficient: pura file memory me load nahi karte.

Example: file stream pipe

```
import { createReadStream, createWriteStream } from "fs";
createReadStream("big.mp4").pipe(createWriteStream("copy.mp4
"));
```

Q7. Buffer kya hota hai?

Ans: Buffer binary data ko represent karta hai (images, files). It’s fixed-size memory chunk outside V8 heap, Node ke low-level I/O me kaam aata hai.

Q8. Cluster vs Worker Threads?

Ans:

- Cluster: multiple Node processes (each with its own event loop) sharing same port behind a master—scale across CPU cores.
- Worker Threads: same process ke andar threads for CPU-intensive tasks (image processing, crypto).

Q9. Process vs Thread difference Node context me?

Ans: Process independent memory space, thread same process memory share karta hai. Node default single thread JS, par background me libuv thread pool hota hai (I/O). CPU-bound ke liye worker threads.

Q10. process.nextTick(), setImmediate(), Promise microtasks ordering?

Ans: Priority: **microtasks (nextTick > Promises)** run before next event loop phase; `setImmediate` check phase me run hota hai; `setTimeout` timers phase me. Overuse of `nextTick` starve kar sakta hai loop ko.

Q11. Error handling best practices?

Ans:

- Async/await ke sath try/catch
- Centralized error middleware (Express)
- Domain-specific messages, no internal stack expose in prod
- Promise rejections handle karo (`process.on('unhandledRejection')`)
- Validation (Joi/Zod) before business logic

Q12. Package.json ke key fields ka role?

Ans: `name`, `version`, `main/exports`, `scripts`, `dependencies`, `devDependencies`, `type`, `engines`. CI/CD, tooling aur module resolution ke liye critical.

Q13. NPM vs Yarn vs PNPM?

Ans: Sab package managers hain. PNPM hard-links symlink store use karta hai → disk efficient, fast installs. Yarn PnP ho sakta hai node_modules ke bina. NPM default, v9+ me performance better.

Q14. Security basics in Node apps?

Ans:

- Dependencies audit (`npm audit`)
- Env secrets (.env + vault)
- Rate limiting, helmet headers
- Validate inputs, sanitize (NoSQL/SQL injection se bacho)
- Avoid eval, serialize safely
- CSRF protection where relevant

Q15. Logging kaise karte ho production me?

Ans: Structured logs (JSON) with **pino**/winston, correlation IDs, request/response meta, log levels (info/warn/error), centralized aggregation (ELK, Loki), sampling for high-traffic.

Q16. Graceful shutdown kya hota hai?

Ans: SIGTERM/SIGINT par new requests stop, in-flight complete, DB/disposables close, then exit. Container orchestration (K8s) friendly.

Example:

```
process.on('SIGTERM', async () => {
  server.close(() => process.exit(0));
});
```

Q17. Rate limiting & throttling difference?

Ans: Rate limiting = max requests per time-window; throttling = speed control (responses delay). Abuse, DDoS aur cost control ke liye zaroori.

Q18. Caching strategies Node backend me?

Ans:

- In-memory (LRU, Map) for hot keys
- Redis for distributed cache
- Cache-aside (read-through), write-through, TTLs, stampede protection (locking)
- HTTP caching headers (ETag, Cache-Control)

Q19. JWT kaise kaam karta hai?

Ans: JWT = signed token (header.payload.signature). Server stateless auth: verify signature + claims (exp, iat). Store in **httpOnly cookies** (XSS resistant) + CSRF defense (SameSite/CSRF token) according to flow.

Q20. File uploads best practice?

Ans: Streaming to object storage (S3/Cloudinary) via pre-signed URLs, virus scanning, size/type validation, avoid storing on app disk, backpressure handle karo.

Q21. Background jobs kaise design karte ho?

Ans: Queue (BullMQ, RabbitMQ), workers for async tasks (emails, webhooks, reports). Retry policy (exponential backoff), idempotency keys, dead-letter queues.

Q22. API versioning strategies?

Ans: URI-based (`/v1`), header-based (`Accept: application/vnd.app.v2+json`), or resource evolution via backward-compatible changes. Deprecation policy + docs.

Express.js (Q23–Q30)

Q23. Express.js kya hai?

Ans: Minimalist web framework for Node—routing, middleware pipeline, request/response helpers. Performance + ecosystem (middlewares) iski strength.

Q24. Middleware kya hota hai Express me?

Ans: Middleware = function `(req,res,next)` jo request/response ko process karta hai (auth, logging, validation). Order matters (pipeline).

Example:

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

Q25. Router ka use kyun aur kaise?

Ans: Route modularization ke liye—feature-wise split.

Example:

```
import { Router } from "express";
const user = Router();
user.get('/:id', getUser);
app.use('/users', user);
```

Q26. Error handling middleware kaise likhte ho?

Ans: 4-args signature `(err, req, res, next)`. Centralize errors, map to HTTP codes, hide internals in prod.

Example:

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

Q27. Validation kaise karte ho?

Ans: Joi/Zod/express-validator. Validate **before** controller logic. Send 400 with helpful error messages; never trust client input.

Q28. Auth in Express – session vs JWT?

Ans:

- Session (server-stored, cookie id) — good for web, revocable.
- JWT (stateless) — good for APIs/mobile; token revocation tricky (use blacklist/short TTL + rotation). Choose per use-case.

Q29. Performance tweaks in Express?

Ans: gzip/br (reverse proxy), HTTP/2 behind nginx, caching headers, avoiding heavy sync ops, pooling DB connections, cluster mode/PM2, compression & helmet carefully (measure!).

Q30. Testing Express APIs?

Ans: Unit (Jest), integration (supertest), contract (OpenAPI + Dredd), E2E (Playwright). Test pyramid: fast unit > some integration > few E2E. Mock external deps.

Database Integration (MongoDB + Mongoose concepts inside) (Q31–Q35)

Q31. Connection management best practices?

Ans: Single shared DB client (singleton) app lifecycle ke sath, connection string via env, retry logic, timeouts, health checks, close on shutdown.

Example (pseudo):

```
import mongoose from 'mongoose';
await mongoose.connect(process.env.MONGO_URI, { maxPoolSize:
10 });
```

Q32. Schema design principles?

Ans: Access patterns se drive karo (query-first). Denormalize where it helps (documents embed), but avoid massive docs. Use proper indexes. Keep documents bounded. Design for growth (sharding keys mindful).

Q33. Indexes kya hote hain aur kab lagate ho?

Ans: Indexes query ko fast banate hain by creating lookup structures. Create on high-selectivity fields used in filters/sorts. Beware: writes slower + storage cost. Composite index order matters (prefix rule).

Q34. Transactions aur atomicity kaise ensure karte ho?

Ans: Multi-document transactions use karo jab zarurat ho (financial ops). Otherwise design idempotent operations + eventual consistency. Retryable writes, unique constraints to avoid duplicates.

Example (txn skeleton):

```
const session = await mongoose.startSession();
await session.withTransaction(async () => {
  await Model1.create([{...}], { session });
  await Model2.updateOne({ ... }, { $inc: { count: 1 }},
{ session });
});
session.endSession();
```

Q35. Pagination & filtering best practices for APIs?

Ans:

- **Cursor-based** pagination (stable, scalable) > offset for large datasets.
- Limit/Max-limit safeguards (e.g., 100).
- Filter whitelists (allowed fields), sort indexes aligned, return metadata (nextCursor).
- Cache hot queries; avoid N+1.

Cursor example (concept):

```
const limit = Math.min(+req.query.limit || 20, 100);
const cursor = req.query.cursor; // last _id or sort key
const q = cursor ? { _id: { $gt: cursor } } : {};
const items = await
Coll.find(q).sort({_id:1}).limit(limit+1);
const nextCursor = items.length > limit ? items[limit-
1]._id : null;
```

Bonus practical Express + DB example (combined)

Request validation + controller + repo separation (concept):

```
// route
user.post('/', validate(createUserSchema), ctrl.create);

// controller
export async function create(req, res, next) {
  try {
    const user = await userRepo.insert(req.body);
    res.status(201).json(user);
  } catch (e) { next(e); }
}
```

Q36. Middleware kya hota hai Express.js me aur kyun important hai?

Ans:

Middleware ek function hai jo **request aur response ke beech** execute hota hai. Ye modular way provide karta hai jaise logging, authentication, validation aur error handling.

- **Order matter karta hai**: middleware pipeline top-down execute hoti hai.
- **Global middleware:** `app.use()` → har request pe run hoga.
- **Route-specific middleware:** sirf selected route pe run hoga.

Example:

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

app.get("/dashboard", authMiddleware, (req, res) => {
  res.send("Dashboard Data");
});
```

Q37. Large Express.js project ka structure kaise rakhein?

Ans:

Large projects ke liye modular architecture best hai:

- `routes/` → saare route definitions
- `controllers/` → business logic
- `models/` → Mongoose schemas
- `middlewares/` → auth, validation, logging
- `utils/` → helper functions
- `config/` → DB connections, env setup

Benefits: maintainable, scalable aur testable.

Q38. Errors ko Express me kaise handle karte hain?

Ans:

- Centralized error middleware `(err, req, res, next)` use karen.
- Client errors (400s) aur server errors (500s) alag handle karein.
- Production me stack trace show na karein.

Example:

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({ message: err.message
|| "Internal Server Error" });
});
```

- Async routes me `try/catch` ya `express-async-handler` use karo.

Q39. Request validation Express me kaise karte hain?

Ans:

Validation ensure karta hai ki data safe aur correct ho. Libraries: **Joi**, **Zod**, **express-validator**.

Example with Joi:

```
import Joi from 'joi';

const schema = Joi.object({
  name: Joi.string().min(3).required(),
  email: Joi.string().email().required(),
});

app.post("/users", (req, res, next) => {
  const { error } = schema.validate(req.body);
  if (error) return res.status(400).json({ message:
error.details[0].message });
  next();
});
```

- Validation hamesha **DB operation ke pehle** karein.
- Ye NoSQL injection aur bad data se bachata hai.

Q40. JWT authentication Node.js me kaise implement karte hain?

Ans:

JWT (JSON Web Token) ek **stateless authentication** method hai.

Steps:

1. User login → server JWT generate kare with **secret key** + payload (user id, role).
2. Client token store kare (httpOnly cookie ya localStorage).
3. Client token bheje `Authorization: Bearer <token>` header me.
4. Middleware verify kare token aur set kare `req.user`.

Example:

```
import jwt from 'jsonwebtoken';

const authMiddleware = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(401).json({ message: "No
token" });
  try {
    const decoded = jwt.verify(token,
process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch {
    return res.status(401).json({ message: "Invalid
token" });
  }
};
```

- Security tips: short TTL, refresh tokens, httpOnly cookies, secure secrets.

Q41. Authentication aur Authorization me difference kya hai?

Ans:

- **Authentication:** verify kare user identity (login, token, session).
- **Authorization:** check kare user permissions (roles, access levels).
- Flow: Authentication first → Authorization next.

Example:

```
if(req.user.role !== "admin") return
res.status(403).json({ message: "Forbidden" });
```

Q42. CORS kya hai aur Express me kaise handle karte hain?

Ans:

CORS → Cross-Origin Resource Sharing. Browser security policy hai jo restrict karta hai cross-origin requests.

Solution:

```
import cors from 'cors';
app.use(cors({
  origin: "http://localhost:3000",
  methods: ["GET", "POST"],
  credentials: true
}));
```

- Postman me ye issue nahi aata, browser me hota hai.
- Dev me allow localhost, production me restrict origin.

Q43. MongoDB connection Node.js me kaise karte hain (Mongoose)?

```
import mongoose from 'mongoose';
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log("DB connected"))
  .catch(err => console.error("DB connection error:", err));
```

- Connection singleton rakhein, environment variables use karein.
- Reconnect aur error handling properly setup karein.

Q44. Mongoose Schema aur Model kya hai?

Ans:

- **Schema:** defines structure, types, validation.
- **Model:** interface DB ke operations ke liye.

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  createdAt: { type: Date, default: Date.now }
});
const User = mongoose.model("User", userSchema);
```

Q45. Population Mongoose me kya hota hai?

- Population allow karta hai references from ek collection to dusre collection.

```
const postSchema = new mongoose.Schema({
  title: String,
  author: { type: mongoose.Schema.Types.ObjectId, ref:
"User" }
});
Post.find().populate("author").exec((err, posts) =>
console.log(posts));
```

- Related data easily fetch karne ke liye.

Q46. CRUD operations MongoDB + Mongoose me kaise?

- **Create:** `Model.create()`
- **Read:** `Model.find()`, `Model.findById()`
- **Update:** `Model.updateOne()`, `Model.findByIdAndUpdate()`
- **Delete:** `Model.deleteOne()`, `Model.findByIdAndDelete()`

```
const user = await User.create({ name: "Aman", email:
"a@b.com" });
```

Q47. Indexing MongoDB me kya hota hai?

- Index query ko fast banata hai.
- Types: single-field, compound, text, TTL.

```
User.collection.createIndex({ email: 1 });
```

- Trade-off: faster reads, slower writes, more storage.

Q48. Transactions MongoDB me kaise use karte hain?

```
const session = await mongoose.startSession();
await session.withTransaction(async () => {
  await User.create([{ name: "Aman" }], { session });
  await Post.updateOne({ _id: postId }, { $inc: { comments:
1 } }, { session });
});
session.endSession();
```

- Multi-document atomicity ke liye.

Q49. Pagination MongoDB me kaise implement karte hain?

- **Offset-based:** `.skip().limit()` → simple
- **Cursor-based:** _id ya timestamp → scalable

```
const limit = 10;
const cursor = req.query.cursor; // last _id
const query = cursor ? { _id: { $gt: cursor } } : {};
const users = await
User.find(query).sort({_id:1}).limit(limit+1);
```

Q50. File uploads Express me kaise handle karte hain?

- Use **multer** middleware
- Stream files → S3/Cloudinary
- Validate size & type, avoid memory overload

```
import multer from "multer";
const upload = multer({ dest: "uploads/" });
app.post("/upload", upload.single("file"), (req, res) =>
res.send("Uploaded"));
```

Q51. API testing Postman / Thunder Client me kaise karte hain?

- Method select kare: GET, POST, PUT, DELETE
- URL add kare, headers/body configure kare
- Response status aur body check kare
- Collections + env variables use karein large testing ke liye

Q52. Authentication API testing Postman me kaise karein?

- Login API → token lein
- `Authorization: Bearer <token>` header me use karein
- Protected routes access karein
- Role-based testing token ke sath

Q53. Rate limiting Express me kaise implement karein?

```
import rateLimit from "express-rate-limit";
const limiter = rateLimit({
  windowMs: 1 * 60 * 1000,
  max: 100,
  message: "Too many requests, try later"
});
app.use("/api", limiter);
```

- Prevent DoS attacks, abuse, ensure fair usage.

Q54. Passwords Node.js apps me kaise secure karein?

- Use **bcrypt** with salt
- Never store plain text

```
import bcrypt from "bcrypt";
const hash = await bcrypt.hash("password123", 10);
```

- Login ke time `bcrypt.compare()` use karo

Q55. CORS issues development me kaise fix karein?

- **cors middleware** use karein
- Allow frontend origin, `credentials: true` for cookies

Q56. Sync vs Async code Node.js me?

- **Sync:** blocking → one by one execution
- **Async:** non-blocking → event loop multiple tasks handle karta hai

Q57. Streams Node.js me kya hai aur kyun use hota hai?

- Large data ko chunks me handle karna → memory efficient
- Types: Readable, Writable, Duplex, Transform
- Use: file upload/download, video streaming

Q58. Clustering Node.js me kya hai?

- Multi-core CPUs utilize karna via worker processes
- Har process ka apna event loop, same port share

```
import cluster from "cluster";
import os from "os";
if(cluster.isMaster){
  os.cpus().forEach(()=>cluster.fork());
}
```

Q59. Aggregation MongoDB me kya hai?

- Data transformation aur analytics pipeline
- Stages: `$match`, `$group`, `$sort`, `$project`
- Example: group users by city & count

Q60. Environment variables Node.js me kaise handle karein?

- Use **dotenv**
- Keep secrets out of repo
- Example: `process.env.MONGO_URI`, `process.env.JWT_SECRET`

Q61. REST vs GraphQL?

- REST → multiple endpoints, fixed response
- GraphQL → single endpoint, client defines fields
- Pros & cons: scalability, flexibility

Q62. WebSockets Node.js me kaise implement karein?

- Real-time bidirectional communication
- Libraries: `ws`, `socket.io`
- Use: chat apps, live dashboards, notifications

Q63. Role-Based Access Control (RBAC) kaise implement karein?

- User roles assign kare: admin, editor, user
- Middleware check kare `req.user.role`
- Unauthorized → 403

Q64. API versioning kaise handle karein?

- URI-based: `/v1/users`
- Header-based: `Accept: application/vnd.app.v1+json`
- Backward compatibility ke liye

Q65. Express.js apps secure kaise karein?

- Helmet → security headers
- Input validation & sanitization

- HTTPS + secure cookies
- Rate limiting
- Avoid unsafe code & eval

Q66. Logging Node.js me kaise implement karein?

- Libraries: **winston**, **pino**
- Structured logs (JSON), levels: info, warn, error
- Production → centralized logging: ELK, Loki

Q67. Node.js performance tips?

- Clustering / worker threads
- Async/await properly
- DB indexing & caching
- Stream large data
- Compression & HTTP2

Q68. File uploads production me kaise handle karein?

- Stream files to S3/Cloudinary
- Validate size/type
- Avoid local storage
- Pre-signed URLs for security

Q69. Node.js debug kaise karein?

- `console.log()` / `console.error()`

- Node Inspector: node `--inspect`
- VSCode debugger
- Postman/Thunder Client API testing

Q70. Deploy Node.js + Express + MongoDB app kaise karein?

- Server: Heroku, Render, AWS EC2, Vercel
- Database: MongoDB Atlas
- Env variables setup: DB URI, JWT secrets
- CI/CD pipelines for automated deployment