Express.js Interview Questions & Answers

Complete Set: Basic (15) • Hard (15) • Ultra-Advanced (20)

Basic Express.js Questions (15)

Q: What is Express.js and why is it used?

A: Express.js is a minimal and flexible Node.js web framework used for building web applications and APIs. It simplifies routing, middleware integration, and handling HTTP requests.

Q: How do you install and set up Express.js?

A: You can install Express using npm (`npm install express`) and create a simple server using `const app = express(); app.listen(3000);`.

Q: What are middleware functions in Express?

A: Middleware are functions that have access to request, response, and next middleware. They are used for logging, authentication, error handling, etc.

Q: Difference between application-level and router-level middleware?

A: Application-level middleware applies to the entire app (`app.use()`), while router-level middleware is bound to an Express router (`router.use()`).

Q: How do you define routes in Express.js?

A: Routes are defined using methods like `app.get('/path', callback)` or `app.post('/path', callback)`.

Q: What is the purpose of the next() function in middleware?

A: The `next()` function passes control to the next middleware in the stack.

Q: How do you serve static files in Express.js?

A: You can serve static files using `app.use(express.static('public'))`.

Q: What is the difference between GET and POST methods in Express?

A: GET is used for retrieving data, while POST is used for sending data to the server.

Q: How do you handle JSON and URL-encoded data in Express?

A: Use `app.use(express.json())` for JSON and `app.use(express.urlencoded({extended:true}))` for form data.

Q: What is the role of app.use() in Express.js?

A: `app.use()` is used to register middleware functions that execute for all HTTP requests.

Q: How do you create a simple REST API with Express.js?

A: By defining routes such as `app.get()`, `app.post()`, `app.put()`, and `app.delete()` with handlers.

Q: What is the difference between app.get() and app.post()?

A: `app.get()` handles GET requests (data retrieval), while `app.post()` handles POST requests (data submission).

Q: How do you handle query parameters in Express.js?

A: Query parameters can be accessed using `req.query.paramName`.

Q: What is the use of express.json() and express.urlencoded()?

A: They are built-in middleware for parsing incoming request bodies.

Q: How do you handle 404 errors in Express applications?

A: By adding a middleware at the end: `app.use((req,res)=>{res.status(404).send('Not Found');});`.

Hard Express.js Questions (15)

Q: How do you implement error-handling middleware in Express.js?

A: Define middleware with 4 arguments `(err, req, res, next)` to catch and process errors.

Q: What are route parameters in Express and how are they different from query parameters?

A: Route parameters are part of the URL (`/user/:id`) and accessed via `req.params.id`, while query parameters are after `?` in the URL and accessed via `req.query`.

Q: How do you organize routes in large Express applications?

A: By using `express.Router()` to modularize routes and separating them into different files.

Q: How do you connect Express.js with a database (e.g., MongoDB)?

A: Use libraries like `mongoose` or `mongodb` driver and integrate inside Express routes.

Q: What is the difference between synchronous and asynchronous middleware in Express?

A: Synchronous middleware runs sequentially, while async middleware uses promises or async/await to handle asynchronous tasks.

Q: How do you handle file uploads in Express.js?

A: By using middleware like `multer` for handling multipart/form-data file uploads.

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

A: CORS allows cross-origin requests. Enable it with `const cors=require('cors'); app.use(cors());`.

Q: How do you implement authentication in Express.js?

A: By using sessions, JWT tokens, or OAuth with middleware like `passport.js`.

Q: What is the difference between session-based and token-based authentication?

A: Session-based authentication stores user data on the server, while token-based authentication (JWT) stores data in the client and validates via tokens.

Q: How do you protect routes in Express applications?

A: By adding authentication middleware before route handlers.

Q: What is Helmet.js and how does it help in securing Express apps?

A: Helmet.js adds HTTP headers to protect against common security vulnerabilities like XSS, clickjacking, etc.

Q: How do you implement rate limiting in Express.js?

A: Use libraries like `express-rate-limit` to restrict repeated requests from the same IP.

Q: How do you use environment variables in Express applications?

A: By using `dotenv` package to load variables from `.env` file into `process.env`.

Q: How do you implement logging in Express.js?

A: By using middleware like `morgan` or custom logging middleware.

Q: How do you test an Express.js application?

A: By using testing frameworks like Jest, Mocha, or Supertest for HTTP endpoint testing.

Ultra-Advanced Express.js Questions (20)

Q: How does Express handle the request-response cycle?

A: Express handles requests by passing them through middleware and route handlers until a response is sent.

Q: Explain how middleware chaining works in Express.

A: Each middleware executes in sequence, and `next()` passes control to the next middleware.

Q: What is the difference between global error handling and route-specific error handling?

A: Global error handling is done with a single middleware at the app-level, while route-specific errors are handled inside route handlers.

Q: How do you optimize Express.js for performance?

A: By enabling caching, using gzip compression, clustering, and minimizing middleware usage.

Q: What are the best practices for structuring large Express.js projects?

A: Using MVC architecture, separating routes, controllers, services, and configuration files.

Q: How does Express handle asynchronous errors?

A: By wrapping async functions in try-catch or using libraries like `express-async-errors`.

Q: What are sub-applications in Express.js?

A: Sub-apps are mini Express applications mounted inside a parent app using `app.use('/subapp', subapp)`.

Q: How do you implement clustering in an Express application?

A: By using Node.js `cluster` module or PM2 to utilize multiple CPU cores.

Q: What is the difference between Express.js and Koa.js?

A: Express provides middleware-based architecture, while Koa is more lightweight and uses async/await heavily.

Q: How do you integrate GraphQL with Express.js?

A: By using `express-graphql` or `apollo-server-express` middleware.

Q: How does Express.js handle streaming data?

A: By using Node.js streams (`req` and `res` are streams) for handling large files or live data.

Q: How do you integrate WebSockets with Express applications?

A: By using libraries like `socket.io` or `ws` alongside Express.

Q: What are the advantages and disadvantages of using Express.js for enterprise apps?

A: Advantages: simplicity, middleware ecosystem, flexibility. Disadvantages: minimal structure, not opinionated.

Q: How do you scale Express.js applications horizontally?

A: By deploying behind load balancers, using clustering, and containerization (Docker/Kubernetes).

Q: How do you handle memory leaks in Express apps?

A: By profiling memory usage, avoiding global variables, and using monitoring tools like PM2.

Q: What is the difference between middleware and controllers in Express?

A: Middleware handles requests/responses before reaching controllers, while controllers contain business logic.

Q: How do you secure sensitive data (like API keys) in Express applications?

A: By using environment variables, secret managers, and avoiding hardcoding credentials.

Q: What are some common security vulnerabilities in Express.js and how do you prevent them?

A: Common issues: XSS, CSRF, SQL Injection. Prevent by using Helmet, input validation, parameterized queries.

Q: How do you implement internationalization (i18n) in Express.js?

A: By using libraries like `i18n` or `express-translate`.

Q: How do you monitor and debug performance issues in Express applications?

A: By using tools like PM2, New Relic, or custom logging and profiling middleware.