Backend Developer Interview Questions & Answers

Q: Node.js: What is the Event Loop?

A: The Event Loop allows Node.js to handle non-blocking I/O operations. It offloads tasks to the system kernel whenever possible, allowing for asynchronous execution of code. Example: Handling multiple HTTP requests in a single-threaded server.

Q: Node.js: Difference between process.nextTick(), setImmediate(), and setTimeout()

A: process.nextTick(): Executes before I/O events.
setImmediate(): Executes on the next iteration of the event loop.
setTimeout(): Executes after a minimum threshold (delay).

Q: Node.js: Handling uncaught exceptions

A: Use process.on('uncaughtException') for global error handling, but prefer using structured try-catch or promise-based error handling.

Q: Express.js: Middleware Functions

A: Middleware functions have access to req, res, and next(). Used for logging, authentication, error handling. Example: app.use(authMiddleware).

Q: Express.js: Application Structure

A: Use MVC or feature-based folder structure for large projects. E.g., /controllers, /models, /routes, /middlewares, /utils.

Q: Express.js: Error Handling

A: Special middleware catches errors: app.use((err, req, res, next) => res.status(500).send('Something broke!'));

Q: MongoDB: find() vs aggregate()

A: find(): Simple queries.
aggregate(): Complex data transformation, e.g., grouping, filtering, and calculating averages.

Q: MongoDB: Indexing Importance

A: Indexes improve query performance by allowing fast data lookups. Example: db.users.createIndex({username: 1}) speeds up username searches.

Q: MongoDB: Schema Design Best Practices

A: Design around your query patterns. Embed data for read-heavy, reference for write-heavy or frequently updated data.

Q: Node.js: Streams

A: Streams allow reading and writing data piece by piece. Useful for large file operations without loading the whole file into memory.

Q: Node.js: Clustering

A: Allows Node.js to utilize multiple CPU cores for scaling. Uses cluster module to spawn child processes.

Q: Node.js: Callback Hell and Solutions

A: Occurs when callbacks are nested, making code hard to read. Avoid using Promises or async/await.

Q: Express.js: Security Practices

A: Use helmet, cors, and express-rate-limit to secure Express apps. Example: app.use(helmet());

Q: Express.js: Routing

A: Defines the endpoint and method handlers, e.g., app.get('/users', handler). Follows REST principles.

Q: Express.js: app.use() vs app.get()

A: app.use() applies middleware globally, app.get() applies logic for GET requests only.

Q: MongoDB: Embedded vs Referenced

A: Embedded: Store related data in the same document for read-heavy scenarios.
Referenced: Use ObjectId for linking documents, ideal for frequently updated data.

Q: MongoDB: Replica Sets

A: Replica Sets provide redundancy and high availability. Secondaries sync data from primary and promote if primary fails.

Q: MongoDB: Transactions

A: MongoDB supports ACID transactions to maintain data integrity across multiple collections or documents.