Complete Node.js Topics and Practice Questions

Core Node.js Fundamentals

1. Node.js Introduction

- What is Node.js?
- Node.js vs Browser JavaScript
- Node.js Architecture
- V8 JavaScript Engine
- Event-Driven, Non-blocking I/O
- Installation and Setup
- Node.js REPL
- Running Node.js Applications
- Node.js vs Other Backend Technologies

2. Node.js Environment

- Global Objects (global, process, console, Buffer)
- Command Line Arguments
- Environment Variables
- Process Object
- OS Module
- Path Module
- URL Module
- Timers (setTimeout, setInterval, setImmediate)

3. Modules and Module System

- CommonJS Modules
- module.exports vs exports

- require() Function
- Module Caching
- Core Modules vs Local Modules vs Third-party Modules
- ES6 Modules (import/export)
- Module Resolution Algorithm
  https://medium.com/outbrain-engineering/node-js-module-resolution-af46715784ef
- Circular Dependencies

4. File System Operations

- Synchronous vs Asynchronous File Operations
- Reading Files (readFile, readFileSync)
- Writing Files (writeFile, writeFileSync)
- File Operations (copy, rename, delete)
- Directory Operations
- File Streams
- Working with CSV and JSON Files

5. HTTP Module

- Creating HTTP Server
- Handling Requests and Responses
- HTTP Methods (GET, POST, PUT, DELETE)
- Request and Response Objects
- Status Codes
- Headers
- Query Parameters
- URL Routing
- Serving Static Files

6. NPM (Node Package Manager)

- package.json Structure
- Installing Dependencies
- Development vs Production Dependencies
- NPM Scripts
- Semantic Versioning
- Package Lock File
- NPM Commands
- NPM vs Yarn

7. Asynchronous Programming

- Callbacks
- Callback Hell
- Promises
- Promise.all(), Promise.race()
- Async/Await
- Error Handling in Async Code
- Event Loop
- Blocking vs Non-blocking Operations

8. Event-Driven Programming

- EventEmitter Class
- Creating Custom Events
- Event Listeners
- Event Propagation
- Removing Event Listeners
- Once vs On Methods
- Error Events

9. Streams

- Readable Streams
- Writable Streams
- Transform Streams
- Duplex Streams
- Pipe Method
- Stream Events
- Backpressure
- Stream Performance

10. Buffer

- Buffer vs String
- Creating Buffers
- Buffer Methods
- Buffer Encoding
- Buffer and Streams
- Memory Management with Buffers

Web Development with Express.js

11. Express.js Fundamentals

- Express Installation and Setup
- Creating Express Applications
- Request and Response Objects
- Routing
- Route Parameters
- Query Strings
- HTTP Methods in Express

12. Middleware

- Built-in Middleware
- Third-party Middleware
- Custom Middleware
- Application-level Middleware
- Router-level Middleware
- Error Handling Middleware
- Middleware Execution Order

14. Static Files and Assets

- express.static Middleware
- Serving CSS, JS, Images
- Asset Organization
- File Upload Handling
- Multer for File Uploads

15. Express Router

- Creating Modular Routes (Nested, Prefixes)
- Router Middleware

Database Integration

16. Working with Databases

- SQL vs NoSQL Databases
- Database Connection Patterns
- Connection Pooling
- Database Configuration
- Environment-based Configuration

17. MongoDB Integration

- MongoDB Connection with Mongoose
- Schema Definition
- Model Creation
- CRUD Operations
- Query Methods
- Population
- Aggregation Pipeline
- Indexing
- Schema Validation

18. MySQL/PostgreSQL Integration

- Database Drivers
- Connection Setup
- Prepared Statements
- Transactions
- Query Building
- ORM vs Raw Queries
- Migration Scripts

19. Database Security

- SQL Injection Prevention
- Data Validation
- Input Sanitization
- Password Hashing
- Database Encryption

Authentication and Security

20. Authentication Strategies

- Session-based Authentication
- Token-based Authentication
- JWT (JSON Web Tokens)
- OAuth Integration
- Passport.js
- Local Authentication
- Social Media Authentication

21. Security Best Practices

- HTTPS Implementation
- CORS (Cross-Origin Resource Sharing)
- Helmet.js for Security Headers
- Rate Limiting
- Input Validation
- XSS Prevention
- CSRF Protection
- Security Auditing

22. Authorization

- Role-Based Access Control (RBAC) (User Roles)
- Permissions based , (permissions)
- Protecting Routes (adding RBAC, permissions)
- Middleware for Authorization

23. Error Handling

- Error Types in Node.js

- Try-Catch Blocks
- Error-First Callbacks
- Promise Error Handling
- Async/Await Error Handling
- Global Error Handlers
- Custom Error Classes

24. Performance Optimization

- Code Profiling

  measuring aspects like execution time, memory usage, and CPU utilization. This process enables developers to pinpoint performance issues and make informed decisions about code improvements

  E.g console.time, console.dir
- Memory Management: heap
- CPU Usage Optimization
- Database Query Optimization

25. Testing

- Unit Testing with Jest (write simple function with 3-4 test case)

26. Debugging

- Node.js Debugger
- VS Code Debugging
- Logging Strategies
- Memory Leak Detection

27. Build Tools and Development

- Nodemon for Auto-restart

- Build Scripts
- Code Linting (ESLint)
- Code Formatting (Prettier)

29. WebSockets

- WebSocket Protocol
- Socket.IO Implementation
- Real-time Communication
- Broadcasting Messages
- Rooms and Namespaces
- Authentication with Socket.IO
- Scaling WebSocket Applications

31. RESTful APIs

- REST Principles
- HTTP Methods and Status Codes
- Resource Naming Conventions
- API Versioning
- API Documentation

33. API Security

- API Authentication
- Rate Limiting
- Input Validation
- CORS Configuration
- API Monitoring

35. Message Queues

- Redis for Queuing

- Job Processing
- Background Tasks

36. Caching

- In-Memory Caching
- Redis Caching

Node.js Ecosystem

37. Popular Libraries and Frameworks

- Express.js
- NestJS
- Hapi.js
- Socket.IO
- Mongoose
- Sequelize

39. Monitoring and Logging

- Winston for Logging
- Morgan for HTTP Logging
- Logging Errors

Project Architecture & Structure

- Common folder layout:

```
routes/       → API routes
controllers/  → Business logic
models/       → Database schemas
middlewares/  → Request/response handlers
utils/        → Helper functions
```

- Keep code **modular** and separated by concern.

Node.js Practice Questions

Environment and Setup

1. Create a simple "Hello World" Node.js application
2. Use command line arguments in a Node.js script
3. Access environment variables
4. Create a basic HTTP server
5. Read and display system information using OS module

File System Operations

1. Read a text file asynchronously and display its content
2. Write data to a file and handle errors
3. Copy a file from one location to another
4. List all files in a directory
5. Create a directory structure programmatically
6. Write, Read and parse a JSON file
7. Stream a large file to avoid memory issues

Modules

1. Create a custom module and export functions
2. Use require() to import built-in modules
3. Create a package.json and add dependencies
4. Handle module caching and circular dependencies

HTTP and Web Servers

1. Create a basic HTTP server that responds to different routes
2. Handle different HTTP methods (GET, POST, PUT, DELETE)
3. Parse query parameters and URL parameters
4. Serve static files from a directory
5. Create a simple REST API with CRUD operations
6. Handle file uploads using streams

Express.js Applications

1. Create an Express.js application with multiple routes
2. Implement middleware for logging requests
3. Create a middleware for authentication
4. Build a CRUD API for a blog system
5. Implement error handling middleware
6. Handle file uploads using form-data payload
7. Implement session-based authentication

Database Integration

1. Connect to MongoDB and perform CRUD operations
2. Create Mongoose schemas and models
3. Implement user registration and login with password hashing
4. Design database relationships (one-to-many, many-to-many)

5. Implement pagination and count for large datasets
6. Create database migration scripts
7. Implement full-text search functionality

Asynchronous Programming

1. Convert callback-based code to Promises
2. Implement async/await for multiple API calls
3. Handle errors in asynchronous operations
4. Use Promise.all() for parallel operations
5. Implement a retry mechanism for failed operations
6. Create a rate limiter using async patterns
7. Implement a job queue system

Authentication and Security

1. Implement JWT-based authentication
2. Create middleware for protected routes
3. Implement password reset functionality
4. Add role-based authorization
5. Implement login with Google/Facebook
6. Create API rate limiting
7. Implement CORS for API security
8. Add input validation and sanitization

Performance

1. Implement clustering for multi-core utilization
2. Implement caching with Redis
3. Profile and optimize memory usage
4. Implement database connection pooling

5. Implement graceful shutdown for applications

Real-time Applications

1. Implement presence system (online/offline status)

Real-World Project Ideas

E-commerce Backend

1. Product catalog management
2. Shopping cart functionality
3. Order processing system
4. Payment integration
5. Inventory management
6. User reviews and ratings
7. Recommendation engine
8. Analytics dashboard

Core Node.js API Libraries

Library	Purpose / Usage
Express.js	Core API framework — routing, middleware, HTTP handling.
Mongoose	MongoDB ODM: schema definitions, queries, validation.
JWT (jsonwebtoken)	Token-based authentication.
bcryptjs	Password hashing for authentication systems.
Joi	Schema-based request validation.
Socket.io	Real-time events and WebSocket communication.
Bull	Job queue processing with Redis.
PM2	Production process manager for Node.js apps.
Multer	File upload handling (multipart/form-data).
CORS	Cross-Origin Resource Sharing control.
Helmet	HTTP security headers.
Winston / Morgan	Logging (application logs & HTTP requests).
Axios	HTTP client for outbound API calls.
Passport.js	Authentication middleware for complex login flows.
Stripe	Payment processing integration.
Twilio	SMS and messaging services.
Firebase	Push notifications & authentication.
Swagger	Auto-generated API documentation & testing.
Nodemailer	Sending emails from backend.
Redis	Caching, session management, pub/sub.
Sharp	Image manipulation and compression.
MinIO	S3-compatible cloud storage.
Google APIs	Integration with Google services like Drive, Gmail.
OpenAI / Groq	AI API integrations (if required by project).

Ref: https://github.com/goldbergyoni/nodebestpractices