1. What is Node.js?

- Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser. It allows developers to use JavaScript for server-side scripting.

2. What is npm?

- npm stands for Node Package Manager. It is the default package manager for Node.js, used to install, share, and manage dependencies in JavaScript projects.

3. What is a callback function in Node.js?

- A callback function is a function passed as an argument to another function, which is then invoked inside the outer function to complete some kind of asynchronous operation. Callbacks are commonly used in Node.js for handling asynchronous tasks like reading files or making network requests.

4. Explain the event loop in Node.js.

- The event loop is the core mechanism of Node.js for handling asynchronous operations. It allows Node.js to perform non-blocking I/O operations despite being single-threaded. The event loop continuously checks for pending tasks, executes them, and then processes the next event in the event queue.

5. What is the difference between setImmediate() and setTimeout() in Node.js?

- `setImmediate()` executes the callback function immediately after the current event loop cycle, while `setTimeout()` schedules the callback to be executed after a specified delay, regardless of the event loop cycle.

6. How do you handle errors in Node.js?

- Errors in Node.js can be handled using try-catch blocks for synchronous code and callback functions with error parameters for asynchronous code. Additionally, Node.js provides the `process.on('uncaughtException')` event to handle uncaught exceptions globally.

7. What is the purpose of package.json in Node.js projects?

- The package.json file contains metadata about the project, such as its name, version, dependencies, scripts, and other configurations. It is used by npm to manage project dependencies and scripts.

8. Explain the concept of streams in Node.js.

- Streams are objects in Node.js that allow reading from or writing to a continuous flow of data. They provide an efficient way to handle large volumes of data by processing it in chunks rather than loading it all into memory at once. Streams can be readable, writable, or both.

9. What is middleware in the context of Express.js?

- Middleware functions in Express.js are functions that have access to the request, response, and next middleware function in the application's request-response cycle. They can modify the request or response objects, end the request-response cycle, or call the next middleware function in the stack.

10. How do you create a basic HTTP server in Node.js?

```javascript
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, World!');
});

server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

Sure, here are some more Node.js interview questions:

11. What is Express.js?

- Express.js is a web application framework for Node.js, designed to build web applications and APIs. It provides a set of features for routing, middleware, and handling HTTP requests and responses.

12. How do you install Express.js in a Node.js project?

- Express.js can be installed via npm using the following command:

```
npm install express
```

13. Explain the difference between `require()` and `import` in Node.js.

- `require()` is a CommonJS module system function used to import modules in Node.js. `import` is an ES6 feature for module import. While Node.js supports `require()`, it also supports `import` when using the `--experimental-modules` flag or with the `.mjs` file extension.

14. What is middleware chaining in Express.js?

- Middleware chaining refers to the process of chaining multiple middleware functions together using the `use()` method in Express.js. This allows developers to apply middleware to specific routes or to the entire application in a sequential manner.

15. How do you handle file uploads in Node.js?

- File uploads in Node.js can be handled using middleware such as `multer`. Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It can be integrated into Express.js applications to handle file uploads.

16. Explain the concept of routing in Express.js.

- Routing in Express.js refers to the process of defining endpoints or routes for handling incoming HTTP requests. Routes are defined using HTTP methods (GET, POST, PUT, DELETE, etc.) and URL patterns, and they specify the callback functions to be executed when a request matches the defined route.

17. What is Promise in Node.js?

- A Promise is an object representing the eventual completion or failure of an asynchronous operation. It allows you to handle asynchronous operations more easily by providing methods for handling success (`then()`) and failure (`catch()`), and chaining asynchronous operations together.

18. How do you handle CORS (Cross-Origin Resource Sharing) in Express.js?

- CORS in Express.js can be handled using the `cors` middleware. The `cors` middleware is used to enable Cross-Origin Resource Sharing by setting the appropriate HTTP headers in the response to allow requests from specified origins.

19. What is clustering in Node.js?

- Clustering in Node.js refers to the ability to spawn multiple instances of a Node.js process to take advantage of multi-core systems. It allows Node.js applications to scale better by distributing the workload across multiple CPU cores.

20. How do you handle authentication in Node.js applications?

- Authentication in Node.js applications can be handled using middleware such as Passport.js or by implementing custom authentication logic using JSON Web Tokens (JWT), sessions, or OAuth.

Certainly! Here are some more Node.js interview questions:

21. What is the purpose of the `util` module in Node.js?

- The `util` module in Node.js provides utility functions that are commonly used for debugging, error handling, and other miscellaneous tasks. For example, it includes functions for working with objects, converting values, and formatting strings.

22. Explain the difference between synchronous and asynchronous code execution in Node.js.

- Synchronous code execution occurs sequentially, where each statement is executed one after another, blocking the execution until the current operation completes. Asynchronous code execution allows multiple operations to be performed concurrently without blocking, enabling non-blocking I/O operations in Node.js.

23. What is the purpose of the `process` object in Node.js?

- The `process` object in Node.js provides information about the current Node.js process, such as environment variables, command-line arguments, memory usage, and exit codes. It also allows you to control the Node.js process, for example, by exiting the process or listening for events.

24. How do you debug Node.js applications?

- Node.js applications can be debugged using built-in debugging tools such as the Node.js debugger (`node inspect`), debugging flags (`--inspect` or `--inspect-brk`), or using third-party debuggers like Visual Studio Code's debugger or Chrome DevTools.

25. Explain the concept of non-blocking I/O in Node.js.

- Non-blocking I/O in Node.js allows the server to handle multiple concurrent connections efficiently without blocking the execution of other operations. Instead of waiting for I/O operations to complete, Node.js continues executing other code, and once the I/O operation finishes, a callback function is invoked to handle the result.

26. What are the global objects in Node.js?

- Some of the global objects in Node.js include `global`, `process`, `console`, `Buffer`, and `module`. These objects are available in all modules and can be accessed without requiring them explicitly.

27. How do you handle environment variables in Node.js applications?

- Environment variables in Node.js applications can be accessed using the `process.env` object. They can be set using command-line arguments, configuration files, or by using environment variable management tools like dotenv.

28. What is the purpose of the `cluster` module in Node.js?

- The `cluster` module in Node.js allows you to create child processes (workers) that share the same server port. It helps in utilizing multiple CPU cores efficiently by distributing incoming connections among the worker processes.

29. Explain the concept of garbage collection in Node.js.

- Garbage collection in Node.js is the process of automatically reclaiming memory occupied by objects that are no longer in use. Node.js uses the V8 JavaScript engine, which employs a generational garbage collector to manage memory efficiently.

30. How do you handle asynchronous errors in Node.js applications?

- Asynchronous errors in Node.js applications can be handled using `try-catch` blocks for synchronous code and by passing error-first callbacks or using `Promise.catch()` for asynchronous code. Additionally, you can use tools like `async/await` or `try-catch` with `await` for cleaner error handling in asynchronous code.

Of course! Here are some more Node.js interview questions:

31. What is event-driven programming in Node.js?

- Event-driven programming in Node.js is a programming paradigm where the flow of the program is determined by events such as user actions, I/O operations, or system events. Node.js uses an event-driven architecture, where certain objects (e.g., EventEmitter) emit events, and listeners (callbacks) are registered to handle those events asynchronously.

32. How do you handle file system operations in Node.js?

- File system operations in Node.js are handled using the `fs` module, which provides functions for interacting with the file system. Operations such as reading files, writing files, creating directories, and deleting files are performed using functions like `fs.readFile`, `fs.writeFile`, `fs.mkdir`, and `fs.unlink`, respectively.

33. What is the purpose of the `path` module in Node.js?

- The `path` module in Node.js provides utilities for working with file paths and directory paths. It allows you to manipulate file paths in a platform-independent manner, resolve relative paths, join paths, and extract path components.

34. Explain the difference between `setInterval()` and `setTimeout()` in Node.js.

- `setInterval()` is a function that repeatedly executes a callback function at specified intervals, while `setTimeout()` is a function that executes a callback function after a specified delay. `setInterval()` continues to execute the callback function at regular intervals until it is cleared using `clearInterval()`, while `setTimeout()` executes the callback function only once.

35. What are streams in Node.js and how are they useful?

- Streams in Node.js are objects that allow you to read from or write to a continuous flow of data in chunks. They are useful for handling large amounts of data efficiently, as they allow you to process data piece by piece without loading it all into memory at once. Streams can be readable, writable, or both, and they are used for various I/O operations such as file I/O, network communication, and data processing.

36. What is the purpose of the `crypto` module in Node.js?

- The `crypto` module in Node.js provides cryptographic functionality, including cryptographic hashing, encryption, decryption, and other cryptographic operations. It allows you to generate secure hashes, create digital signatures, encrypt data using various algorithms, and perform other security-related tasks.

37. Explain the difference between `process.nextTick()` and `setImmediate()` in Node.js.

- `process.nextTick()` schedules a callback function to be executed asynchronously on the next iteration of the event loop, immediately after the current operation completes. `setImmediate()` schedules a callback function to be executed asynchronously on the next iteration of the event loop, but after I/O events are processed. In general, `process.nextTick()` has higher priority than `setImmediate()`.

38. What is the purpose of the `url` module in Node.js?

- The `url` module in Node.js provides utilities for URL parsing and formatting. It allows you to parse URLs into their components (protocol, host, path, query parameters, etc.) and format URL components into a valid URL string.

39. How do you perform unit testing in Node.js applications?

- Unit testing in Node.js applications can be performed using testing frameworks like Mocha, Jest, or Jasmine, along with assertion libraries like Chai or Node.js built-in `assert` module. Test suites and test cases are written to verify the behavior of individual units (functions, modules, etc.) of the application.

40. What are child processes in Node.js and how do you create them?

- Child processes in Node.js are separate instances of the Node.js process that can be created to execute other programs or run code in parallel. They are useful for running CPU-intensive tasks, executing shell commands, or running legacy code. Child processes can be created using the `child_process` module, which provides functions like `spawn()`, `exec()`, and `fork()`.

Certainly! Here are some more Node.js interview questions:

41. What is the purpose of the `Buffer` class in Node.js?

- The `Buffer` class in Node.js is used to handle binary data, such as reading from or writing to streams, file operations, and network communication. It represents a fixed-size chunk of memory allocated outside the V8 JavaScript engine's heap, allowing Node.js to work with binary data efficiently.

42. Explain the difference between `exports` and `module.exports` in Node.js.

- `exports` is a shortcut reference to `module.exports`, which is an object that is initially empty. When you assign a value to `exports`, you are modifying the reference, but when you assign a value to `module.exports`, you are replacing the entire exports object. `module.exports` is what is actually exported from a module.

43. How do you handle environment-specific configuration in Node.js applications?

- Environment-specific configuration in Node.js applications can be managed using environment variables, configuration files, or external configuration management tools. Typically, configuration values are stored in separate files (e.g., JSON, YAML) for each environment (development, production, etc.) and loaded dynamically based on the current environment.

44. What is the purpose of the `os` module in Node.js?

- The `os` module in Node.js provides operating system-related utility functions, allowing you to get information about the current operating system, such as CPU architecture, network interfaces, memory usage, and operating system platform.

45. Explain the difference between `require()` and `import` for module loading in Node.js.

- `require()` is a CommonJS module loading system used in Node.js to import modules dynamically. `import` is an ES6 feature for module loading, primarily used in modern JavaScript environments. While Node.js supports `require()`, it also supports `import` when using the `--experimental-modules` flag or with the `.mjs` file extension.

46. How do you handle sessions in Node.js applications?

- Sessions in Node.js applications can be managed using middleware like `express-session`, which stores session data on the server and assigns a unique session identifier (session ID) to each client. Session data can be stored in memory, databases, or other storage mechanisms, and it can be used to maintain user authentication, store user preferences, and track user interactions.

47. What is the purpose of the `http` module in Node.js?

- The `http` module in Node.js provides functionality for creating HTTP servers and clients. It allows you to handle HTTP requests, route requests to appropriate handlers, and send HTTP responses. Additionally, it provides utilities for working with HTTP headers, status codes, and cookies.

48. How do you implement caching in Node.js applications?

- Caching in Node.js applications can be implemented using in-memory caching mechanisms like `node-cache` or `memory-cache`, external caching solutions like Redis or Memcached, or by leveraging CDN (Content Delivery Network) caching for static assets. Caching helps improve application performance by storing frequently accessed data in memory or external storage for faster retrieval.

49. Explain the purpose of the `net` module in Node.js.

- The `net` module in Node.js provides functionality for creating TCP servers and clients. It allows you to establish TCP (Transmission Control Protocol) connections, listen for incoming connections, and communicate over TCP sockets. The `net` module is used for low-level networking operations in Node.js applications.

50. What are the advantages of using Node.js for building web applications?

- Some advantages of using Node.js for building web applications include its non-blocking, event-driven architecture, which allows for high concurrency and scalability; its ability to share code between the client and server using JavaScript; its rich ecosystem of npm packages and modules; and its support for building real-time, data-intensive applications.

Certainly! Here are some additional Node.js interview questions:

51. What is middleware in the context of Express.js?

- Middleware in Express.js is a function that has access to the request object (`req`), response object (`res`), and the next middleware function in the application's request-response cycle. It can perform tasks such as modifying request or response objects, terminating the request-response cycle, or passing control to the next middleware function.

52. How do you handle routing in an Express.js application?

- Routing in Express.js involves defining routes for handling HTTP requests. Routes are defined using HTTP methods (`GET`, `POST`, `PUT`, `DELETE`, etc.) and URL patterns, and they specify the callback functions to be executed when a request matches the defined route.

53. What is JWT (JSON Web Token) and how is it used in Node.js applications?

- JWT is a compact, URL-safe token format used for securely transmitting information between parties as a JSON object. In Node.js applications, JWTs are commonly used for authentication and authorization by encoding user information into a token, which can be verified and decoded on subsequent requests.

54. How do you handle database operations in Node.js applications?

- Database operations in Node.js applications can be handled using database-specific libraries such as `mysql`, `mongodb`, `sequelize`, or `mongoose`, depending on the type of database (SQL or NoSQL) being used. These libraries provide APIs for connecting to databases, executing queries, and performing CRUD operations on data.

55. What are promises in Node.js and how do they work?

- Promises in Node.js are objects representing the eventual completion or failure of an asynchronous operation. They allow you to handle asynchronous code more elegantly by chaining `.then()` and `.catch()` methods to handle success and error cases, respectively. Promises help avoid callback hell and make asynchronous code more readable.

56. Explain the concept of middleware error handling in Express.js.

- Middleware error handling in Express.js involves defining error-handling middleware functions with four parameters (`err`, `req`, `res`, `next`). These middleware functions are executed when an error occurs during the request-response cycle. They can log errors, customize error responses, or perform cleanup tasks.

57. What is clustering in Node.js and how does it work?

- Clustering in Node.js involves spawning multiple instances of a Node.js process to utilize the full processing power of a multi-core system. It works by creating a cluster of worker processes, where each worker process runs on a separate CPU core and shares the same server port to handle incoming requests.

58. How do you handle file uploads in an Express.js application?

- File uploads in Express.js applications can be handled using middleware such as `multer` or `formidable`. These middleware parse the incoming form data containing files, store the files temporarily on the server, and provide access to the file data, allowing you to process and save the files as needed.

59. Explain the purpose of the `child_process` module in Node.js.

- The `child_process` module in Node.js provides functionality for spawning child processes, executing shell commands, and interacting with external processes. It allows you to run CPU-intensive tasks in parallel, execute shell scripts, or interact with system-level processes from within a Node.js application.

60. What are the different deployment options for Node.js applications?

- Node.js applications can be deployed using various options, including traditional servers (e.g., Apache, Nginx), cloud platforms (e.g., AWS, Google Cloud, Microsoft Azure), containerization platforms (e.g., Docker, Kubernetes), serverless platforms (e.g., AWS Lambda, Google Cloud Functions), or using PaaS (Platform as a Service) providers that offer Node.js hosting.

Certainly! Here are more Node.js interview questions:

61. What is the purpose of the `EventEmitter` class in Node.js?

- The `EventEmitter` class in Node.js is used to implement the observer pattern, allowing objects to emit named events that cause registered listeners to be invoked. It provides methods like `on()`, `emit()`, and `once()` to add listeners, emit events, and listen for events, respectively.

62. How do you handle database migrations in Node.js applications?

- Database migrations in Node.js applications can be managed using migration libraries like `migrate`, `knex`, or `sequelize`. These libraries allow you to define migration files that specify changes to the database schema over time, such as creating tables, altering columns, or seeding data.

63. What is WebSockets and how do you implement WebSocket communication in Node.js applications?

- WebSockets is a communication protocol that provides full-duplex communication channels over a single TCP connection, allowing real-time bidirectional communication between clients and servers. In Node.js applications, WebSocket communication can be implemented using libraries like `ws` or `socket.io`, which provide APIs for creating WebSocket servers and handling WebSocket connections.

64. How do you handle authentication using Passport.js in an Express.js application?

- Authentication using Passport.js in an Express.js application involves configuring Passport.js strategies (e.g., local strategy, OAuth strategy) to authenticate users using different authentication mechanisms. Passport.js middleware is then used to authenticate requests by verifying user credentials and managing user sessions.

65. What is middleware error handling in Express.js and how is it different from regular error handling?

- Middleware error handling in Express.js involves defining error-handling middleware functions with four parameters (`err`, `req`, `res`, `next`). These middleware functions are executed when an error occurs during the request-response cycle. Middleware error handling allows you to centralize error handling logic and separate it from regular route handling middleware.

66. How do you handle sessions in Express.js applications and what are session stores?

- Sessions in Express.js applications can be managed using middleware like `express-session`, which stores session data on the server and assigns a unique session identifier (session ID) to each client. Session stores are storage mechanisms (e.g., memory store, database store) used to store session data persistently across requests.

67. Explain the purpose of the `assert` module in Node.js.

- The `assert` module in Node.js provides functions for writing assertions, allowing you to test whether values meet certain conditions and throw errors if they don't. It is commonly used for writing unit tests and performing runtime assertions in Node.js applications.

68. How do you handle environment-specific configuration in Node.js applications using dotenv?

- Environment-specific configuration in Node.js applications can be managed using the `dotenv` module, which loads environment variables from a `.env` file into `process.env`. `.env` files contain key-value pairs of configuration variables, which are loaded into the Node.js environment during application startup.

69. What is serverless computing and how can you deploy Node.js applications as serverless functions?

- Serverless computing is a cloud computing model where cloud providers dynamically manage the allocation and provisioning of servers, allowing developers to focus on writing code without worrying about server management. Node.js applications can be deployed as serverless functions using platforms like AWS Lambda, Google Cloud Functions, or Azure Functions, which execute code in response to events without provisioning or managing servers.

70. Explain the purpose of the `cluster` module in Node.js and how it helps in scaling Node.js applications.

- The `cluster` module in Node.js allows you to create multiple instances of a Node.js process (workers) that share the same server port. It helps in scaling Node.js applications by utilizing multiple CPU cores efficiently, distributing incoming requests among worker processes, and improving application performance and concurrency.

Certainly! Here are some additional Node.js interview questions:

71. What are the differences between Node.js and traditional web servers like Apache or Nginx?

- Node.js is a runtime environment for executing JavaScript code on the server side, whereas servers like Apache or Nginx are traditional web servers primarily used for serving static files and processing HTTP requests. Node.js is single-threaded and event-driven, making it suitable for handling I/O-heavy tasks and real-time applications.

72. How do you handle form validation in Node.js applications?

- Form validation in Node.js applications can be performed using validation libraries like `express-validator`, which provides middleware for validating and sanitizing user input received via HTTP requests. It allows you to define

validation rules for request parameters and validate incoming data against those rules.

73. Explain the purpose of the `crypto` module in Node.js and how it can be used for encryption and decryption.

- The `crypto` module in Node.js provides cryptographic functionality, including encryption, decryption, hashing, and generating cryptographic keys. It can be used to encrypt sensitive data before storing it in a database or transmitting it over a network, and to decrypt data when it needs to be accessed or processed.

74. What is the purpose of the `util.promisify()` function in Node.js?

- The `util.promisify()` function in Node.js is used to convert callback-based functions into Promise-based functions. It takes a callback-style function as input and returns a new function that returns a Promise, allowing you to use `async/await` syntax or Promise chaining with functions that follow the Node.js callback convention.

75. How do you handle cross-origin resource sharing (CORS) in Express.js applications?

- CORS in Express.js applications can be handled using middleware like `cors`, which allows you to configure CORS policies to specify which origins are allowed to access the server's resources, which HTTP methods are allowed, and which headers can be sent in cross-origin requests. The `cors` middleware automatically adds appropriate CORS headers to HTTP responses.

76. Explain the purpose of the `crypto.createHash()` function in Node.js and how it can be used for generating hash digests.

- The `crypto.createHash()` function in Node.js is used to create a Hash object that can be used to generate hash digests (hash values) for input data using various cryptographic hash algorithms such as MD5, SHA-1, or SHA-256. It takes the name of the hash algorithm as an argument and returns a Hash object that can be used to compute hash digests.

77. What is the purpose of the `process.argv` array in Node.js and how can it be used to access command-line arguments?

- The `process.argv` array in Node.js contains the command-line arguments passed to the Node.js process. The first element (`process.argv[0]`) is the path to the Node.js executable, the second element (`process.argv[1]`) is the path to the script being executed, and subsequent elements are the command-line arguments passed to the script. It can be used to access and parse command-line arguments passed to a Node.js script.

78. How do you handle file uploads in a pure Node.js (without Express.js) application?

- File uploads in a pure Node.js application can be handled using the built-in `http` module to create an HTTP server and the `formidable` or `multiparty` modules to parse incoming form data containing file uploads. You would listen for incoming requests, parse the form data, handle file uploads, and respond accordingly.

79. Explain the purpose of the `crypto.randomBytes()` function in Node.js and how it can be used to generate cryptographically secure random bytes.

- The `crypto.randomBytes()` function in Node.js is used to generate cryptographically secure random bytes of a specified length. It takes the desired number of bytes as an argument and returns a Buffer containing the randomly generated bytes. It is commonly used for generating random cryptographic keys, salts, or initialization vectors.

80. What is the purpose of the `os` module in Node.js and how can it be used to retrieve information about the operating system?

- The `os` module in Node.js provides utilities for interacting with the operating system, allowing you to retrieve information such as CPU architecture, hostname, operating system platform, total memory, and network interfaces. It provides methods like `os.cpus()`, `os.totalmem()`, and `os.networkInterfaces()` for accessing system information.

Absolutely! Here are more Node.js interview questions:

81. What is the difference between `process.nextTick()` and `setImmediate()`?

- `process.nextTick()` queues a function to be executed asynchronously at the end of the current event loop cycle, before any I/O events. `setImmediate()` queues a function to be executed asynchronously on the next iteration of the event loop, after the current I/O events.

82. How can you implement server-side rendering (SSR) in a Node.js application?

- Server-side rendering in a Node.js application involves using a template engine like EJS, Handlebars, or Pug to generate HTML on the server and send it to the client. You render dynamic content within templates using data retrieved from a database or other sources.

83. What is middleware stacking in Express.js?

- Middleware stacking in Express.js involves using the `app.use()` method to add multiple middleware functions to the application's request-response pipeline. Middleware functions are executed sequentially in the order they are

defined, allowing you to perform various tasks such as logging, authentication, and error handling.

84. How do you handle file downloads in a Node.js application?

- File downloads in a Node.js application can be handled by streaming file data directly to the HTTP response using the `fs.createReadStream()` function to read the file and the `response.pipe()` method to stream the data to the client.

85. What is the purpose of the `http.createServer()` method in Node.js?

- The `http.createServer()` method in Node.js is used to create an HTTP server instance that listens for incoming HTTP requests on a specified port. It takes a callback function as an argument, which is invoked each time a request is received, allowing you to handle the request and send an appropriate response.

86. How do you handle file uploads with progress tracking in a Node.js application?

- File uploads with progress tracking in a Node.js application can be achieved using libraries like `express-fileupload`, which provides middleware to handle file uploads and emits events to track the progress of file uploads.

87. What is the purpose of the `Buffer` class in Node.js and how is it used?

- The `Buffer` class in Node.js is used to represent binary data, allowing you to work with raw binary data directly. It provides methods for encoding and decoding data in various formats, such as ASCII, UTF-8, and Base64.

88. How can you handle database transactions in a Node.js application?

- Database transactions in a Node.js application can be handled using database-specific libraries like `sequelize` or `knex`, which provide APIs for managing transactions using the `transaction` method or `transaction` object. Transactions allow you to perform multiple database operations atomically, ensuring data integrity.

89. What is the purpose of the `crypto.createCipheriv()` function in Node.js?

- The `crypto.createCipheriv()` function in Node.js is used to create a Cipher object that can be used to encrypt data using a specified encryption algorithm and initialization vector (IV). It takes the encryption algorithm, key, and IV as arguments and returns a Cipher object that can be used to encrypt data.

90. How do you handle JSON Web Tokens (JWT) authentication in a Node.js application?

- JWT authentication in a Node.js application involves generating JWTs containing user information on successful login, sending the JWT to the client, and including the JWT in subsequent requests for authentication. Middleware can be used to verify the JWT and extract user information from it.

Certainly! Here are some more Node.js interview questions:

91. Explain the concept of event loop in Node.js.

- The event loop in Node.js is a mechanism that allows Node.js to handle asynchronous operations efficiently. It continuously checks the event queue for pending events, executes event handlers for those events, and processes I/O operations asynchronously, allowing Node.js to perform non-blocking I/O operations.

92. What is the purpose of the `module.exports` object in Node.js?

- The `module.exports` object in Node.js is used to export variables, functions, or objects from a module to make them available for use in other modules. It allows you to encapsulate code within modules and expose only the necessary functionality to other parts of the application.

93. How do you handle errors in asynchronous code in Node.js?

- Errors in asynchronous code in Node.js can be handled using `try-catch` blocks for synchronous code and by using error-first callbacks, promises with `.catch()`, or `async/await` with `try-catch` for asynchronous code. Error handling mechanisms allow you to catch and handle errors gracefully.

94. Explain the purpose of the `cluster` module in Node.js and how it can be used for load balancing.

- The `cluster` module in Node.js allows you to create a cluster of worker processes that share the same server port, enabling load balancing across multiple CPU cores. It helps in improving the performance and scalability of Node.js applications by distributing incoming requests among worker processes.

95. What is the purpose of the `fs` module in Node.js?

- The `fs` module in Node.js provides functionality for interacting with the file system, allowing you to perform operations such as reading from or writing to files, creating directories, deleting files, and more. It provides both synchronous and asynchronous methods for file I/O operations.

96. How do you handle concurrency in Node.js applications?

- Concurrency in Node.js applications can be handled using asynchronous programming techniques such as callbacks, promises, or `async/await`. These

techniques allow you to perform multiple tasks concurrently without blocking the event loop, enabling efficient handling of I/O-bound operations.

97. Explain the purpose of the `http` module in Node.js and how it can be used to create an HTTP server.

- The `http` module in Node.js provides functionality for creating HTTP servers and clients. It can be used to create an HTTP server instance using the `http.createServer()` method, which listens for incoming HTTP requests on a specified port and handles those requests using a callback function.

98. What is the purpose of the `child_process` module in Node.js?

- The `child_process` module in Node.js provides functionality for spawning child processes, executing shell commands, and interacting with external processes. It allows you to run CPU-intensive tasks, execute shell scripts, or run external commands from within a Node.js application.

99. How do you implement a RESTful API in an Express.js application?

- Implementing a RESTful API in an Express.js application involves defining routes for various HTTP methods (GET, POST, PUT, DELETE) and URL patterns to handle CRUD operations on resources. Each route maps to a controller function that handles the request, processes the data, and sends an appropriate response.

100. Explain the purpose of the `net` module in Node.js and how it can be used to create TCP servers.

- The `net` module in Node.js provides functionality for creating TCP servers and clients. It can be used to create a TCP server instance using the `net.createServer()` method, which listens for incoming TCP connections on a specified port and handles those connections using a callback function.

DYNAMIC SYSTEM

T R A I N I N G | R E S E A R C H | D E V E L O P M E N T

LOCATION: BENGALURU | BHUBANESWAR | HYDERABAD

APPLY FOR JOBS: Private jobs|Free Job Registration (dynamicsystemindia.in)