### <https://javascriptcentric.medium.com/top-50-nodejs-interview-questions-and-answers-for-2024-5e460dac7852>

### What is Node.js? Where can you use it?

[Node.js is an open-source](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs), cross-platform [JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript) runtime environment and library to run web applications outside the client’s browser**.**It is used to create server-side web applications.

Node.js is perfect for data-intensive applications as it uses an asynchronous, event-driven model. You can use I/O intensive web applications like video streaming sites. You can also use it for developing: Real-time web applications, Network applications, General-purpose applications, and Distributed systems.

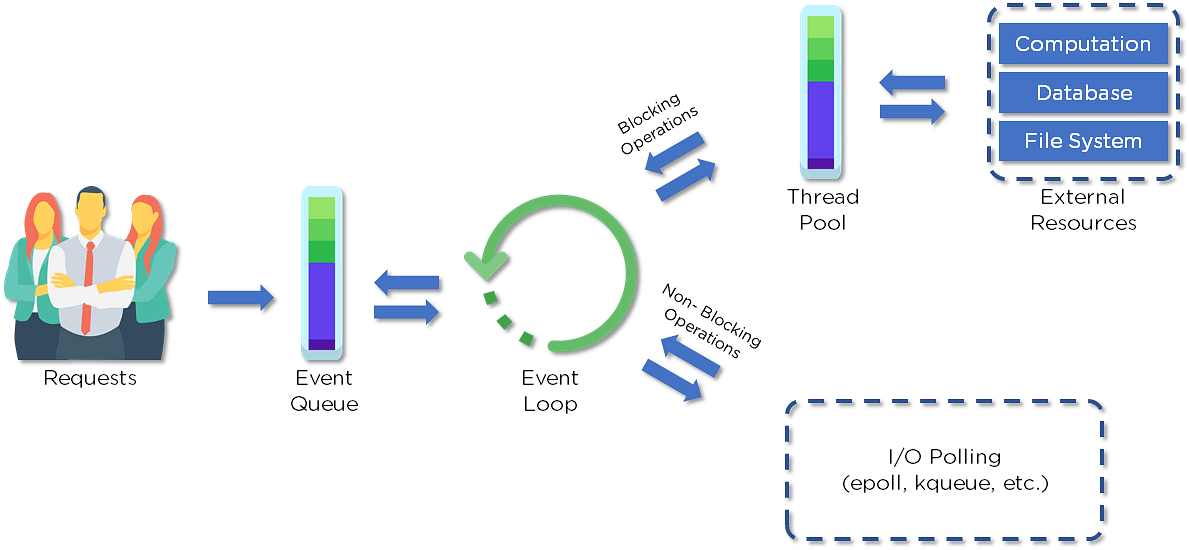
### Why use Node.js?

Node.js makes building scalable network programs easy. Some of its advantages include:

* It is generally fast
* It rarely blocks
* It offers a unified programming language and data type
* Everything is asynchronous
* It yields great concurrency

### How does Node.js work?

A web server using Node.js typically has a workflow that is quite similar to the diagram illustrated below. Let’s explore this flow of operations in detail.



Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking:

* Querying for data
* Deleting data
* Updating the data

Node.js retrieves the incoming requests and adds those to the Event Queue

The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough not to require any external resources

The Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients

A single thread from the Thread Pool is assigned to a single complex request`. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.

Once the task is carried out completely, the response is sent to the Event Loop that sends that response back to the client.

**Is Node.js Single-threaded?**

Yes, Node.js is single-threaded. It uses a single thread to handle requests and relies on asynchronous, non-blocking operations to manage multiple tasks concurrently. However, it can leverage multiple threads internally via the libuv library for tasks like I/O operations, file system handling, and other background tasks.

### Why is Node.js Single-threaded?

Node.js is single-threaded to simplify its design and facilitate asynchronous, non-blocking I/O operations. This approach offers several advantages:

1. **Simplified Concurrency**: Avoids the complexity of multi-threaded programming, such as dealing with thread synchronization and data consistency issues.
2. **Efficient I/O Operations**: Uses an event-driven, non-blocking I/O model, which allows it to handle many concurrent connections without being bogged down by the limitations of thread context switching.
3. **Scalability**: The single-threaded event loop can handle thousands of concurrent connections, making it highly scalable for I/O-bound applications.
4. **Developer Productivity**: Simplifies the development process and error handling by avoiding the pitfalls of multi-threaded programming.

### If Node.js is single-threaded, then how does it handle concurrency?

The Multi-Threaded Request/Response Stateless Model is not followed by the Node JS Platform, and it adheres to the Single-Threaded Event Loop Model. The Node JS Processing paradigm is heavily influenced by the JavaScript Event-based model and the JavaScript callback system. As a result, Node.js can easily manage more concurrent client requests. The event loop is the processing model's beating heart in Node.js.

Node.js handles concurrency through its event-driven, non-blocking I/O model. Here's how it works:

1. **Event Loop**: The core of Node.js is the event loop, which continuously checks for and processes events, such as incoming requests or completed I/O operations.
2. **Callback Functions**: When a request or an I/O operation is initiated, Node.js delegates the task to the system and attaches a callback function to it. The event loop registers these callbacks to be executed once the operations are complete.
3. **Asynchronous Operations**: Node.js uses asynchronous operations, meaning it doesn't wait for an operation to complete before moving on to the next one. This allows Node.js to handle other tasks while waiting for the initial operation to finish.
4. **Worker Threads**: For CPU-intensive tasks, Node.js can use the worker threads module to spawn additional threads. These threads operate in the background and communicate with the main thread via message-passing, allowing Node.js to handle heavy computation without blocking the event loop.
5. **libuv Library**: Under the hood, Node.js uses the libuv library, which provides a thread pool to manage asynchronous I/O operations. This library helps Node.js to offload tasks such as file system operations, DNS resolution, and other I/O-bound tasks to separate threads, ensuring the main event loop remains non-blocked and responsive.

### Explain callback in Node.js.

A callback function is called after a given task. It allows other code to be run in the meantime and prevents any blocking.  Being an asynchronous platform, Node.js heavily relies on callback. All APIs of Node are written to support callbacks.

In Node.js, a callback is a function passed as an argument to another function, which is then executed after the completion of the asynchronous operation. Callbacks are essential in Node.js for handling asynchronous events and tasks without blocking the main execution thread. Here's how they work:

const fs = require('fs');

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) {

console.error('Error reading file:', err);

return;

}

console.log('File contents:', data);

});

console.log('This message is printed first');

**Explanation**

1. **fs.readFile**: The fs.readFile function is called to read the contents of example.txt. It takes the file path, encoding, and a callback function as arguments.
2. **Callback Function**: The callback function (err, data) => { ... } is passed as an argument. It will be executed once fs.readFile completes reading the file.
3. **Asynchronous Nature**: While fs.readFile is reading the file, the main code continues to execute, and console.log('This message is printed first'); is printed immediately.
4. **Handling the Result**: Once the file reading is complete, the callback function is invoked. If there is an error, it is handled inside the callback; otherwise, the file contents are logged to the console.

**Benefits of Callbacks**

* **Non-blocking**: Callbacks allow Node.js to handle I/O operations without blocking the main thread, leading to better performance and responsiveness.
* **Simplified Code Flow**: They help manage the flow of asynchronous operations, ensuring that certain code executes only after specific tasks are completed.

**What is callback hell?**

Callback hell, also known as the "pyramid of doom," refers to a situation in asynchronous programming where callbacks are nested within other callbacks multiple levels deep. This nesting makes the code hard to read, maintain, and debug due to its complicated and indented structure. It's a common issue in Node.js when dealing with multiple asynchronous operations that depend on each other.

### Example of Callback Hell

const fs = require('fs');

// First asynchronous operation

fs.readFile('file1.txt', 'utf8', (err, data1) => {

if (err) {

console.error('Error reading file1:', err);

return;

}

console.log('File 1 contents:', data1);

// Second asynchronous operation

fs.readFile('file2.txt', 'utf8', (err, data2) => {

if (err) {

console.error('Error reading file2:', err);

return;

}

console.log('File 2 contents:', data2);

// Third asynchronous operation

fs.readFile('file3.txt', 'utf8', (err, data3) => {

if (err) {

console.error('Error reading file3:', err);

return;

}

console.log('File 3 contents:', data3);

// And so on...

});

});

});

### Solutions to Callback Hell

To avoid callback hell, you can use modern JavaScript features such as Promises and async/await.

#### Using Promises

Promises provide a way to handle asynchronous operations more cleanly and avoid deeply nested callbacks.

const fs = require('fs').promises;

// Using Promises to avoid callback hell

fs.readFile('file1.txt', 'utf8')

.then(data1 => {

console.log('File 1 contents:', data1);

return fs.readFile('file2.txt', 'utf8');

})

.then(data2 => {

console.log('File 2 contents:', data2);

return fs.readFile('file3.txt', 'utf8');

})

.then(data3 => {

console.log('File 3 contents:', data3);

})

.catch(err => {

console.error('Error:', err);

});

#### Using async/await

The async/await syntax provides an even cleaner way to write asynchronous code that looks more like synchronous code.

const fs = require('fs').promises;

async function readFiles() {

try {

const data1 = await fs.readFile('file1.txt', 'utf8');

console.log('File 1 contents:', data1);

const data2 = await fs.readFile('file2.txt', 'utf8');

console.log('File 2 contents:', data2);

const data3 = await fs.readFile('file3.txt', 'utf8');

console.log('File 3 contents:', data3);

} catch (err) {

console.error('Error:', err);

}

}

readFiles();

### What are the advantages of using promises instead of callbacks?

One downside of using callbacks extensively is "callback hell," where code becomes deeply nested and harder to read. This can be mitigated by using Promises and async/await, which provide a cleaner and more manageable way to handle asynchronous operations in Node.js.

Using Promises instead of callbacks offers several advantages, particularly in terms of code readability, error handling, and maintainability.

* The control flow of asynchronous logic is more specified and structured.
* The coupling is low.
* We've built-in error handling.
* Improved readability.

### How would you define the term I/O?

* The term I/O is used to describe any program, operation, or device that transfers data to or from a medium and to or from another medium
* Every transfer is an output from one medium and an input into another. The medium can be a physical device, network, or files within a system



### 

### Explain the difference between frontend and backend development?

|  |  |
| --- | --- |
| **Front-end** | **Back-end** |
| Frontend refers to the client-side of an application | Backend refers to the server-side of an application |
| It is the part of a web application that users can see and interact with | It constitutes everything that happens behind the scenes |
| It typically includes everything that attributes to the visual aspects of a web application | It generally includes a web server that communicates with a database to serve requests |
| HTML, CSS, JavaScript, AngularJS, and ReactJS are some of the essentials of frontend development | Java, PHP, Python, and Node.js are some of the backend development technologies |

### What is NPM?

NPM stands for Node Package Manager, responsible for managing all the packages and modules for Node.js.

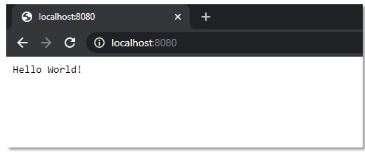
Node Package Manager provides two main functionalities:

* Provides online repositories for node.js packages/modules, which are searchable on search.nodejs.org
* Provides command-line utility to install Node.js packages and also manages Node.js versions and dependencies

### 

### What are the modules in Node.js?

Modules are like JavaScript libraries that can be used in a Node.js application to include a set of functions. To include a module in a Node.js application, use the **require()** function with the parentheses containing the module's name.



Node.js has many modules to provide the basic functionality needed for a web application. Some of them include:

|  |  |
| --- | --- |
| **Core Modules** | **Description** |
| HTTP | Includes classes, methods, and events to create a Node.js HTTP server |
| util | Includes utility functions useful for developers |
| fs | Includes events, classes, and methods to deal with file I/O operations |
| url | Includes methods for URL parsing |
| query string | Includes methods to work with query string |
| stream | Includes methods to handle streaming data |
| zlib | Includes methods to compress or decompress files |

### What is the purpose of the module .Exports?

In Node.js, a module encapsulates all related codes into a single unit of code that can be parsed by moving all relevant functions into a single file. You may export a module with the module and export the function, which lets it be imported into another file with a needed keyword.

### Why is Node.js preferred over other backend technologies like Java and PHP?

Some of the reasons why Node.js is preferred include:

* Node.js is very fast
* Node Package Manager has over 50,000 bundles available at the developer’s disposal
* Perfect for data-intensive, real-time web applications, as Node.js never waits for an API to return data
* Better synchronization of code between server and client due to same code base
* Easy for web developers to start using Node.js in their projects as it is a JavaScript library

### [****What is the difference between Node.js and JavaScript?****](https://www.geeksforgeeks.org/difference-between-node-js-and-javascript)

JavaScript is a scripting language whereas Node.js is an engine that provides the runtime environment to run JavaScript code.

Here we have [difference table between Node.js and JavaScript](https://www.geeksforgeeks.org/difference-between-node-js-and-javascript)

| [**Node.js**](https://www.geeksforgeeks.org/node-js-introduction) | [**JavaScript**](https://www.geeksforgeeks.org/introduction-to-javascript) |
| --- | --- |
| Server-side runtime environment | Client-side scripting language |
| Allows running JavaScript code on server | Primarily used for web development |
| Built on Chrome’s V8 JavaScript engine | Runs in a web browser’s JavaScript engine |
| Enables building scalable network applications | Executes code within a browser environment |
| Provides access to file system and network resources | Limited to browser APIs and capabilities |
| Supports event-driven, non-blocking I/O operations | Executes in a single-threaded event loop |
| Used for building backend APIs, servers, and applications | Utilized for creating interactive web pages and client-side logici |

**Mention the benefits of using Node.js.**

* The APIs in Node.js are asynchronous which means that they do not cause blockers. This happens due to the Node.js server not waiting for the return of data once an API has been called. It moves on to the next API. Instead, there is a notification mechanism that gets the response from the prior API call. Thus, we can also say that Node.js is event-driven.
* Node.js is powered by the Google Chrome V8 JavaScript Engine. This makes code execution in the Node.js library super fast.
* Node.js is based on a single-thread model with event looping. The event mechanism in the Node.js library makes it highly scalable because it prevents blockers on the server as compared to other servers that can only handle limited requests. In Node.js, the single thread program can service copious requests compared to something like Apache HTTP Server.
* Because of the event mechanism in Node.js, there is no buffering. Instead, the data output occurs in chunks.

**Explain global installation of dependencies.**

/npm directory stores the globally installed dependencies. While these dependencies can be used in the Command Line Interface or CLI function of all Node.js libraries, they cannot be directly imported in the Node application with the require() command. If you want to globally install a dependency, you can use the -g flag.

**Explain REPL in Node.js.**

REPL stands for Read Eval Print Loop. REPL performs tasks related to reading, executing, printing, and looping. The server contains several ad-hoc Java statements that REPL can help execute. With REPL, the JavaScript can enter the shell post directly and reliably perform tasks like debugging, testing, and experimenting.

**Define Node.js.**

Node.js is a JavaScript Runtime Environment that is useful in developing server-side applications.

**Is Node.js cross-platform?**

Yes, Node.js is cross-platform and we can run it on Windows, Linux, Unix, and macOS.

**What makes Node.js different?**

It is different from other JavaScript environments because it is asynchronous and event-driven.

**Is Node.js open-source or not?**

Node.js is open-source and cross-platform.

**What can we build with Node.js?**

[Node.js developers](https://www.turing.com/jobs/remote-node-js-developer) can build various applications, including web applications, chat applications, real-time applications, streaming applications, APIs, and desktop applications, etc.

**Does Node.js use JavaScript?**

Node.js is based on JavaScript and it uses the V engine developed by Google. It is used for building server-side applications.

Node.js uses JavaScript through the V8 engine, which is Google's open-source JavaScript engine. Here's how it works:

JavaScript code is written to handle server-side logic, such as handling HTTP requests, accessing databases, and managing file systems.

1. **JavaScript Execution**: The V8 engine compiles JavaScript code directly to native machine code, which allows for fast execution.
2. **Event-Driven Architecture**: Node.js uses an event-driven, non-blocking I/O model. This means it can handle many operations concurrently without waiting for any of them to complete before moving to the next one.
3. **Modules**: Node.js has a module system, where developers can include various built-in modules or third-party modules using require(), making it easy to manage and organize code.
4. **Asynchronous Programming**: JavaScript in Node.js uses callbacks, promises, and async/await to handle asynchronous operations, allowing for efficient and scalable server-side applications.
5. **Single-Threaded but Concurrency-Friendly**: Although Node.js runs on a single thread, it uses an event loop to manage multiple operations concurrently, making it suitable for I/O-heavy tasks.

**Can we run Node.js on Windows?**

Yes, it is possible to run Node.js on Windows.

**Download Node.js**: Go to the [Node.js official website](https://nodejs.org/) and download the Windows installer (.msi) for the LTS version.

**Install Node.js**: Run the installer and follow the prompts, ensuring Node.js is added to your PATH environment variable.

**Verify Installation**: Open Command Prompt or PowerShell and run node -v and npm -v to check the installed versions.

**Run a Node.js Application**:

* Create a file app.js.
* Add console.log('Hello, Node.js on Windows!'); to the file.
* Run node app.js from the Command Prompt or PowerShell in the file's directory to see the output.

**What do you mean by I/O?**

I/O stands for input/output, which helps write and read files and network operations.

**What are the two data types categories in Node.js?**

Node.js supports two categories of data type - primitive and non-primitive.

**Name types of API functions supported by Node.js.**

Asynchronous non-blocking and synchronous blocking are the two different types of API functions that Node.js supports.

1. **Asynchronous Non-Blocking APIs**:

* These APIs allow the program to continue executing other code while waiting for an operation to complete.
* They often use callbacks, promises, or async/await for handling the completion of these operations.
* Examples:
  + fs.readFile(): Reads a file asynchronously.
  + http.get(): Makes an HTTP GET request asynchronously.
  + setTimeout(): Executes a function after a specified delay without blocking the main thread.

1. **Synchronous Blocking APIs**:

* These APIs block the execution of the code until the operation completes.
* They are useful for simple scripts or tasks where performance is not a critical concern.
* Examples:
  + fs.readFileSync(): Reads a file synchronously.
  + require(): Loads a module synchronously.

While asynchronous non-blocking APIs are preferred for most Node.js applications due to their efficiency and scalability, synchronous blocking APIs can be useful in certain scenarios where simplicity is more important than performance.

**Why does Google use V8 for Node.js?**

Google uses V8 for Node.js because it is faster and more efficient. It compiles the JavaScript code directly into machine code.

**Which extension is used to save Node.js files?**

.js extension is used to save Node.js files.

**What is full form of npm?**

Npm stands for the [Node package manager](https://www.turing.com/blog/top-npm-packages-for-node-js-developers/). It is used for installing, updating, and uninstalling packages in your application. It helps to manage dependencies in Node.js applications.

**For what type of applications is Node.js not compatible?**

Node.js is not compatible with CPU-intensive applications.

**What is Node red?**

Node red is a visual programming tool for Node.js that is used to wire hardware devices and online services as part of IoT applications.

**How is operational error different from programming error?**

An operational error occurs naturally and is part of the application flow, while programming errors are referred to as bugs that are caused by developers.

**What is unit testing?**

Unit testing in Node.js is a process of testing individual units of code.

**What is the blocking code?**

Blocking code is code that cannot be executed until the current code is completely executed.

**Are there any disadvantages of Node.js?**

No technology comes without a few disadvantages. Node.js also has a few drawbacks. The main drawback is that Node.js responses can be greatly blocked if an intensive CPU computation is used.

In such cases, the multiple thread options are better but they are sluggish in performance. Moreover, if a relational database is used with Node.js, it behaves strangely, preventing favorable outcomes that the users can be sure of. Since Node.js does not support multiple threads, it is better suited for lightweight applications but not large-scale or heavy applications

**Give some examples of async functions.**

Some examples of async functions are setTimeout(), setInterval(), and process.nextTick()

**Which library provides Node.js with a JavaScript engine?**

The V8 library provides Node.js with the JavaScript engine.

**How is JavaScript different from Node.js?**

JavaScript is a programming language, whereas Node.js is an interpretation and environment for JavaScript. Node.js is used for performing non-blocking operations of any operating system. On the other hand, JavaScript is used for comprehensive application development.

**Can you develop network applications with Node.js?**

Yes, Node.js developers can develop a variety of applications, including network applications.

**What tasks can be done asynchronously with the help of an event loop?**

The tasks include - intensive CPU computation, I/O operations, GUI programming, and database operations can be done asynchronously with the help of an event loop.

**What are security implementations within Node.js?**

The different types of security implementations within Node.js include error handling, authentications and authorization, data sanitization, encryption, and logging and monitoring.

**What is Package.json?**

Present in the root directory of a Node application/module, package.json defines the properties of a package including dependencies, metadata, and configuration options.

**What are the clauses used in promise object in Node.js?**

In JavaScript, a Promise object can have three states:

**Pending**: The initial state of the promise before it is resolved or rejected.

**Fulfilled**: The state of a promise representing a successful operation. This is also sometimes called "resolved."

**Rejected**: The state of a promise representing a failed operation. To create a Promise object, you must pass a function (often called an executor function) to the Promise constructor.  
This function takes two arguments: resolve and reject. These are functions that you call to either fulfill or reject the promise.

**Name some important applications in IT where Node.js can be used.**

One of the main applications where Node.js can be used is building real-time web applications. Apart from this, distributed systems for sub-programming collections can also use Node.js. It can be used in general applications as well as complex network applications. One can also use Node.js to create, read, write, or close server files.

**Is AJAX supported by all browsers?**

Yes, all browsers support AJAX.

**How to obtain the IP address of the user in Node.js?**

We use req.connection.remote address to get the IP address.

**How to install the Node body-parser module?**

Run the command “npm install body-parser” to install the body-parser module and add it to your project's dependencies.

**Explain the function of exit code in Node.js.**

Exit codes are a collection of specific codes whose function is to complete a specific process. Examples of exit codes are fatal error, unused, internal JavaScript evaluation failure, etc.

**What causes server latency and prevents scalability in Node.js?**

Several factors can cause server latency and prevent scalability in Node.js. Some of the most common ones include:

* Blocking I/O: Blocking I/O operations can cause the server to become unresponsive while waiting for I/O operations to complete. This can be especially problematic in Node.js, which is designed to handle many concurrent connections. To avoid this, Node.js provides non-blocking I/O operations.
* Inefficient code: Inefficient code can cause unnecessary processing and slow down the server. This can be caused by poor algorithmic choices, excessive use of synchronous operations, or inefficient data structures.
* Insufficient hardware resources: Insufficient hardware resources, such as CPU, memory, or network bandwidth, can cause the server to become overloaded and unresponsive. This can be especially problematic in high traffic scenarios.
* Improper configuration: Improperly configured servers can cause performance issues. This can be caused by incorrect network settings, improper load balancing, or other misconfigurations.

**How does Node.js convert JavaScript code to C++?**

Node.js uses the Google V8 JavaScript engine to convert JS code to C++.

**Define event programming.**

Event programming is programming paradigm that uses events to trigger actions. An event can be generated by the user, by the system, or by the program itself.

**How is Ajax different from Node.js?**

Ajax is a client-side technology used to make web pages more interactive and dynamic, while Node.js is a server-side technology used to build scalable, high-performance web applications.

**Explain non-blocking in Node.js.**

Non-blocking in Node.js means that the program can continue to execute other code even while waiting for I/O operations to complete.

**In which packages are dependencies stored in Node.js?**

Dependencies are present in the package.json file.

**Define control function in Node.js.**

A control function manages and manipulates the flow of asynchronous code execution.

Node.js is designed to handle asynchronous I/O operations, which means that multiple I/O operations can be executed simultaneously without blocking the execution of other code. However, managing the flow of asynchronous code can be challenging, especially when multiple operations need to be executed in a particular order.

Control functions provide a solution to this problem by allowing developers to define the order in which asynchronous operations should be executed. They can be used to perform tasks such as error handling, callback management, and flow control.

**When do you use modularization in Node.js?**

Modularization in Node.js provides scalability when developing complex applications. The modularization option can be used to execute the import of objects, classes, functions, modules, and external files.

**What is the call-back function used for?**

The call-back function is used to execute a function after a certain event has occurred.

**Explain how blocking is prevented in Node.js.**

Because of the event mechanism in Node.js, a callback function is called every time an event starts. This prevents blocking in Node.js.

**How many layers are there in Node.js application architecture?**

There are three layers in the [application architecture](https://www.turing.com/kb/understanding-the-nodejs-architecture) - API, service, and integration layers.

**Name input arguments for asynchronous queue.**

Two input arguments for asynchronous queue are concurrency value and task function.

**Does Node.js applications buffer data?**

No. Node.js applications do not buffer data.

**Is it possible to run external processes with Node.js?**

It is possible to run external processes with Node.js. This can be done with the help of the child\_process module.

**What is the function of the fs module?**

The fs module is used to create and manipulate files. It also provides an API for interacting with the file system.

**Define os module in Node.js?**

The os module provides a set of tools for interacting with the operating system. It provides an API for getting information about the system including memory, processor, file system, and network interfaces.

**Are duplex streams readable and writable?**

Duplex streams are both readable and writable. This means that they can be used to read data from a source and write data to a destination.

**What is a transform stream?**

A transform stream is a type of stream that can be used to modify or transform the data as it is being read or written.

**What is the meaning of HTTP status code ?**

HTTP status code indicates that the server is unable to process the request. This can be due to several reasons, such as an overloaded server or a network issue.

HTTP status codes are standardized codes returned by web servers to indicate the outcome of an HTTP request made by a client (such as a web browser or API client). They consist of a three-digit number where the first digit specifies the category of the response. Here's a breakdown of the categories and some common status codes:

HTTP status codes are standardized codes returned by web servers to indicate the outcome of an HTTP request made by a client (such as a web browser or API client). They consist of a three-digit number where the first digit specifies the category of the response. Here's a breakdown of the categories and some common status codes:

1. **1xx: Informational**:

**100 Continue**: The server has received the request headers, and the client should proceed to send the request body.

**101 Switching Protocols**: The server is switching protocols as requested by the client.

1. **2xx: Success**:

**200 OK**: The request has succeeded.

**201 Created**: The request has been fulfilled, and a new resource has been created.

**204 No Content**: The server successfully processed the request, but there is no content to send in the response.

1. **3xx: Redirection**:

**301 Moved Permanently**: The resource has been permanently moved to a new URL.

**302 Found**: The resource is temporarily located at a different URL.

**304 Not Modified**: The resource has not been modified since the last request, so the client can use the cached version.

1. **4xx: Client Errors**:

**400 Bad Request**: The server cannot process the request due to client error (e.g., malformed request syntax).

**401 Unauthorized**: Authentication is required and has failed or has not yet been provided.

**403 Forbidden**: The server understands the request but refuses to authorize it.

**404 Not Found**: The requested resource could not be found on the server.

1. **5xx: Server Errors**:

**500 Internal Server Error**: The server encountered an unexpected condition that prevented it from fulfilling the request.

**502 Bad Gateway**: The server, while acting as a gateway or proxy, received an invalid response from the upstream server.

**503 Service Unavailable**: The server is currently unavailable (e.g., due to overload or maintenance).

**Explain routing in the express.**

Routing is a process where you associate HTTP request to a URL path or specific routes. When a request matches one of these routes, a corresponding handling function is used. Routing is a robust mechanism that enables you to define how an application handles incoming requests.

**How to open a file in Node.js?**

A file can be opened in Node.js using the fs.open() method. This method takes two arguments, the path of the file and the flags.

**What is the difference between Angular and Node.js?**

Angular is a front-end web application framework. Node.js is a back-end runtime environment.

**How does Node.js handle concurrency if it is single-threaded?**

Node.js prevents bottlenecks and aids programmers in easily writing the code because of the single-thread model. Internally, there are several POSIX threads for different I/O operations like File, DNS, etc.

So, when Node receives an I/O request, it uses one of these threads for the I/O operation. Once the operation is complete, the result joins the event queue. Because of the event mechanism, the event loop starts after each event, checks the queue, and if Node’s execution stack is free, then the loop adds the queue result to it, thus managing concurrency.

**What is the shortcut for killing a process in Node.js?**

Ctrl + C shortcut is used for terminating processes in Node.js.

**What is a Node Inspector?**

A Node-Inspector is a debugging tool that allows developers to inspect and debug the code of an application through a graphical user interface.

**Is dgram an in-built module?**

The Dgram is an in-built module. It is useful in implementing UDP datagram packets.

**Can we import a buffer class without buffer modules?**

You can import the buffer class without the buffer module.

**Which function is used to fire an event?**

The emit() function is used to fire an event.

**Explain call back.**

In Node.js, a callback function is a function that is passed as an argument to another function and is called back at a later point in time. The purpose of using a callback function is to handle asynchronous operations.

In asynchronous programming, a function that initiates an asynchronous operation returns immediately while the operation continues in the background. When the operation is completed, the callback function is called with the result of the operation.

**Can middleware function execute code?**

Yes, the middleware functions can execute code and they can modify the request or response objects.

**How many types of streams are there in Node.js?**

**Readable**: The readable streams are used to read data from a specific source.

**Writable**: These streams are utilized for writing data to the destination.

**Duplex**: The duplex streams are used for both reading and writing data.

**Transform**: The transform stream enable the data to be transformed while it is being read or written. For example, you can use transform stream for data compression or data encryption as it is transmitted over the network.

**What is the difference between setImmediate() and setTimeout()?**

The difference between setImmediate() and setTimeout() is that setImmediate() will execute the callback function immediately, while setTimeout() will wait for a specified time before executing the callback function.

**What two arguments do async.queue take?**

Task function and concurrency value are the two arguments async.queue takes.

**What is the Null data type in Node.js?**

It is a special data type that only takes one value, i.e., null.

**What is the difference between an event and a callback?**

The difference between an event and a callback is that an event is a mechanism that signals a change or action that represents a part of the program's behaviour, while a callback is a function that takes data and sends it back to the calling function.

**How will you delete a directory?**

To delete a directory, we use fs. rmdir() method.

**Is the value of the symbol data type kept private?**

Yes, it is private and used internally.

**What is an error-first callback?**

Error-first callbacks are used to pass data and identify if an error has occurred.

**What are global objects?**

Global objects are universal objects that are present in every module or file of the application without requiring explicit import statements.

**When does the child process occur?**

In Node.js, a child process occurs when a new process is created using the child\_process module. This module provides functionality to spawn child processes in a similar way to the fork() system call in Unix.

**Can variables defined with let keyword be redeclared?**

No, the let keyword can’t be redeclared and should be declared before use.

**Explain the control flow function.**

A control flow function controls the order in which code is executed. Usually, the code runs from the first line to the last line unless the structures (loops and conditions) changes the control flow.

**What is the function of module.exports?**

module.exports is a Node.js module that allows you to export objects, functions, and variables from your code. You can use it to create modules that can be imported and used in other Node.js programs.

**What is libuv?**

Libuv is a library that provides an event loop and a thread pool for handling asynchronous operations seamlessly. It helps in handling file systems, child processes, files, DNS, etc.

**When are we required to use a cluster module in Node.js?**

It is used when you want to disperse request processing over multiple Node.js processors. Mostly this is done to ramp up the ability to handle more requests per second.

**How does Node.js use cryptography?**

Node.js uses cryptography for a variety of purposes, including password hashing, public key authentication, and encrypting network traffic. Some of the crypto modules used in Node.js are crypto, tls, crypto.createHash(), and crypto.createCipher() among others.

**What are the important features of the npm directory?**

* A package.json file that defines the dependencies for your project
* A Node\_modules directory that contains the modules for your project
* An .npmignore file that defines which files should be ignored while publishing your module

**How to include HTTP server in the Node module?**

You can include an HTTP server in your Node module by using the http module. The http module provides an API for creating and interacting with HTTP servers and clients. It allows you to handle incoming HTTP requests and send HTTP responses, as well as control the server's behavior, such as listening on a specific port and setting various options.

To include an HTTP server in a Node.js module, you can use the built-in http module. Here are the steps to create a basic HTTP server:

const http = require('http'); // **Import the HTTP module**:

const server = http.createServer((req, res) => { // **Create the server**:

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello, World!\n');

});

const PORT = 3000;

server.listen(PORT, () => { // **Listen on a specific port**:

console.log(`Server running at http://localhost:${PORT}/`);

});

**What is the function of npm –g command?**

The npm –g command installs the specific package for your project globally through npm which is the Node.js package manager.

**How to scale a Node.js application?**

You can scale a Node.js application by using a cluster. A cluster is a pool of worker processes that share a single port.

**Clustering**: Utilize Node.js's built-in cluster module to create multiple instances (workers) of your application, taking advantage of multi-core processors.

**Load Balancing**: Distribute incoming traffic across multiple server instances.

Tools: Use tools like Nginx, HAProxy, or cloud-based load balancers (AWS ELB, Google Cloud Load Balancer).

**Microservices Architecture**: Break down the application into smaller, independent services that can be developed, deployed, and scaled independently.

Tools: Docker for containerization, Kubernetes for orchestration.

**Horizontal Scaling**: Add more instances of your application across multiple servers.

Implementation: Use cloud platforms (AWS, Azure, GCP) to easily scale out by adding more instances.

**Caching**: Store frequently accessed data in memory to reduce database load.

Tools: Redis, Memcached.

**Database Optimization**: Optimize database queries, use indexing, and consider database sharding or replication.

Tools: MongoDB sharding, SQL indexing.

**Auto-scaling**: Automatically adjust the number of running instances based on the current load.

Tools: AWS Auto Scaling, Google Cloud AutoScaler.

**What is the process of connecting the output of one stream to another called?**

The process of connecting the output of one stream to another is called piping.

**What’s Punycode in Node.js?**

Punycode is a way of encoding internationalized domain names that use characters outside of the ASCII range.

**What is the purpose of EventEmitter?**

The purpose of EventEmitter is to provide a mechanism for event-driven programming. The EventEmitter class provides a set of methods that allow you to emit events, listen for events, and remove event listeners. When an event is emitted, any listeners that are registered for that event will be invoked. This allows developers to write code that can respond to changes in state, user actions, or other events within the application.

**Explain the child process module in Node.js.**

The child process module allows you to spawn child processes in Node.js. The child process module provides a way to create and control child processes. A child process is a separate instance of the Node.js process that runs concurrently with the parent process.

The child process module allows you to spawn new child processes, execute commands in a shell, and communicate with child processes using IPC (Inter-Process Communication) channels.

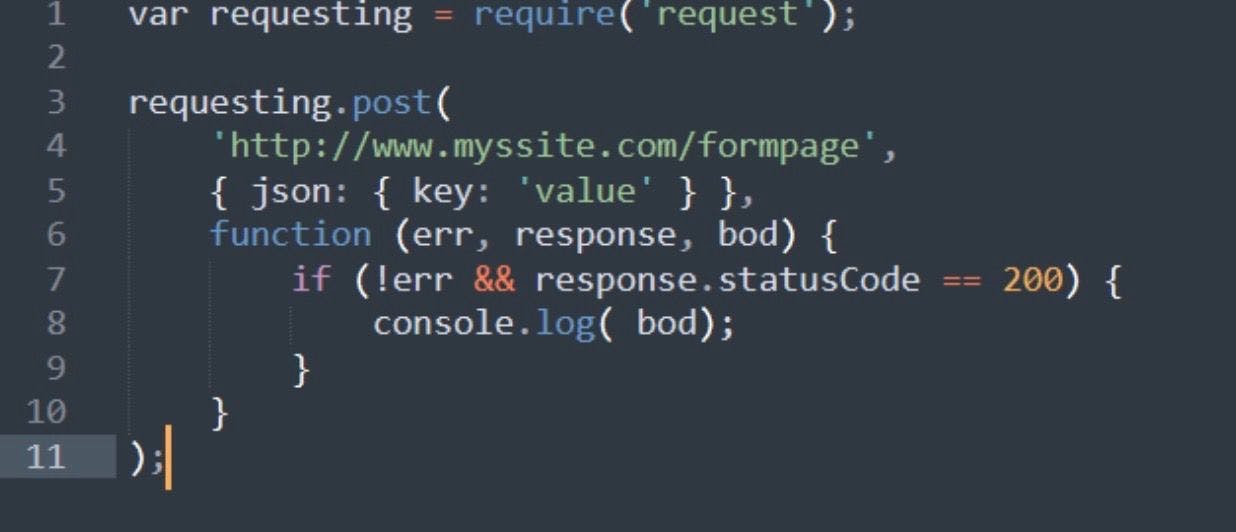
**What does the CharAt() function stand for?**

This method returns a character at a particular index in a string.

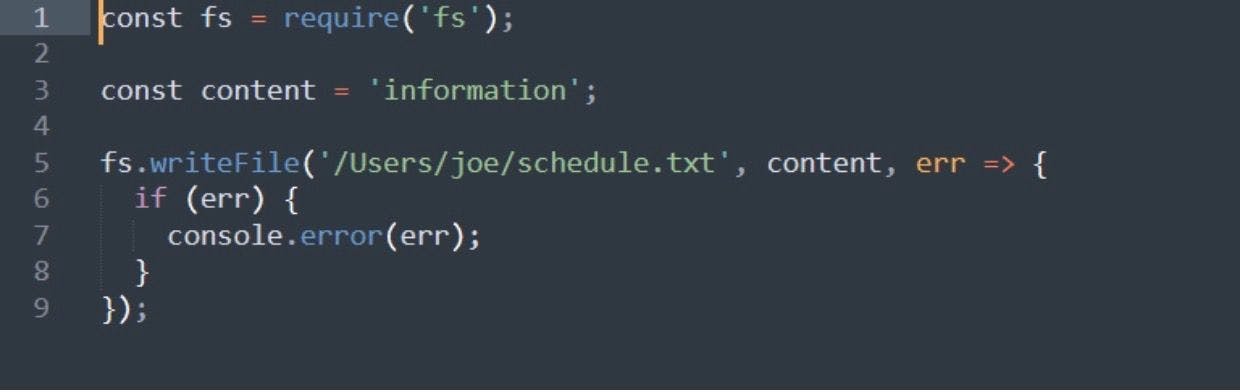
**What is the meaning of the Split() method?**

The Split() function splits the string and returns an array of strings.

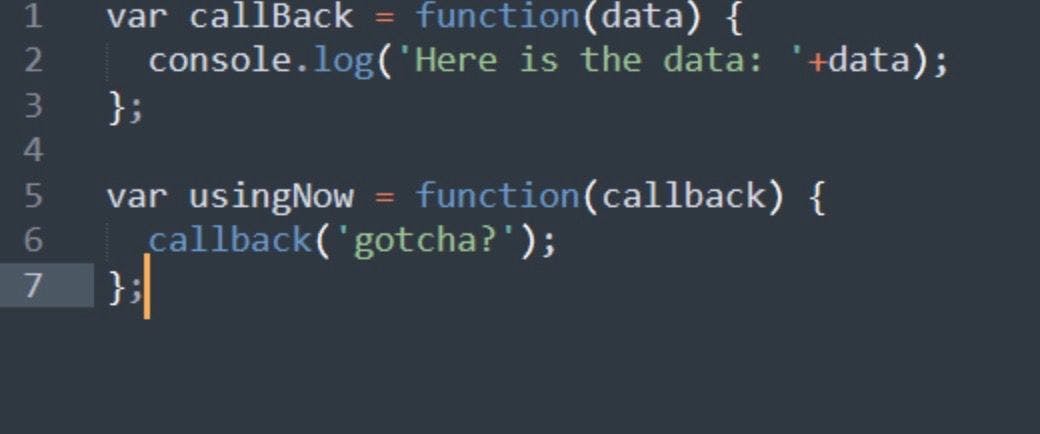
**Write a code to make post requests in Node.js.**



**How to write files with Node.js?**



**Write a callback function in Node.js?**



### Which database is more popularly used with Node.js?

[MongoDB](https://www.simplilearn.com/tutorials/mongodb-tutorial) is the most common database used with Node.js. [It is a NoSQL](https://www.simplilearn.com/rise-of-nosql-and-why-it-should-matter-to-you-article), cross-platform, document-oriented database that provides high performance, high availability, and easy scalability.

### What are some of the most commonly used libraries in Node.js?

There are two commonly used libraries in Node.js:

* [**ExpressJS**](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js)- Express is a flexible Node.js web application framework that provides a wide set of features to develop web and mobile applications.
* **Mongoose** - [Mongoose](https://www.simplilearn.com/tutorials/nodejs-tutorial/nodejs-mongodb) is also a Node.js web application framework that makes it easy to connect an application to a database.

### What are the pros and cons of Node.js?

|  |  |
| --- | --- |
| **Node.js Pros** | **Node.js Cons** |
| Fast processing and an event-based model | Not suitable for heavy computational tasks |
| Uses JavaScript, which is well-known amongst developers | Using callback is complex since you end up with several nested callbacks |
| Node Package Manager has over , packages that provide the functionality to an application | Dealing with relational databases is not a good option for Node.js |
| Best suited for streaming huge amounts of data and I/O intensive operations | Since Node.js is single-threaded, CPU intensive tasks are not its strong suit |

### What is the command used to import external libraries?

The “require” command is used for importing external libraries. For example - **“var http=require (“HTTP”).”**  This will load the HTTP library and the single exported object through the HTTP variable.

Now that we have covered some of the important beginner-level Node.js interview questions let us look at some of the intermediate-level Node.js interview questions.

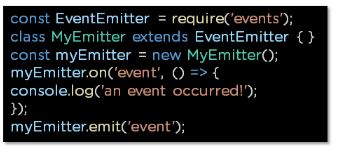
varhttp

### What does event-driven programming mean?

An event-driven programming approach uses events to trigger various functions. An event can be anything, such as typing a key or clicking a mouse button. A call-back function is already registered with the element executes whenever an event is triggered.

### What is an EventEmitter in Node.js?

* EventEmitter is a class that holds all the objects that can emit events
* Whenever an object from the EventEmitter class throws an event, all attached functions are called upon synchronously

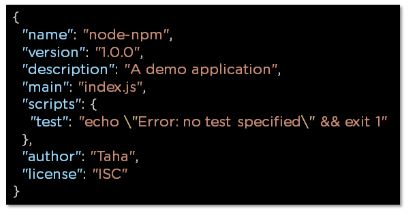


### What is the package.json file?

The package.json file is the heart of a Node.js system. This file holds the metadata for a particular project. The package.json file is found in the root directory of any Node application or module

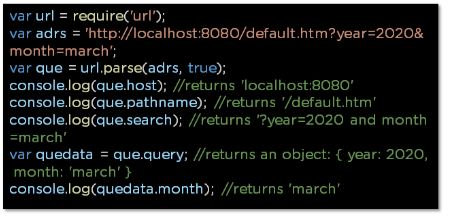
This is what a package.json file looks like immediately after creating a Node.js project using the command: **npm init**

You can edit the parameters when you create a Node.js project.



### How would you use a URL module in Node.js?

The URL module in Node.js provides various utilities for URL resolution and parsing. It is a built-in module that helps split up the web address into a readable format.



### What is the Express.js package?

Express is a flexible Node.js web application framework that provides a wide set of features to develop both web and mobile applications

### How do you create a simple Express.js application?

* The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on
* The response object represents the HTTP response that an Express app sends when it receives an HTTP request



### What are streams in Node.js?

Streams are objects that enable you to read data or write data continuously.

There are four types of streams:

**Readable –** Used for reading operations

**Writable −** Used for write operations

**Duplex −** Can be used for both reading and write operations

**Transform −** A type of duplex stream where the output is computed based on input

### How do you install, update, and delete a dependency?



### How do you create a simple server in Node.js that returns Hello World?



* Import the HTTP module
* Use createServer function with a callback function using request and response as parameters.
* Type “hello world."
* Set the server to listen to port and assign an IP address

### Explain asynchronous and non-blocking APIs in Node.js.

* All Node.js library APIs are asynchronous, which means they are also non-blocking
* A Node.js-based server never waits for an API to return data. Instead, it moves to the next API after calling it, and a notification mechanism from a Node.js event responds to the server for the previous API call

### 

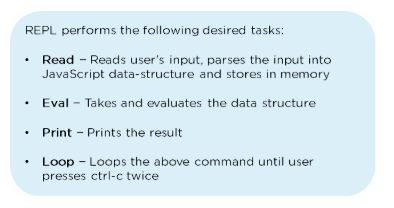
### How do we implement async in Node.js?

As shown below, the async code asks the JavaScript engine running the code to wait for the request.get() function to complete before moving on to the next line for execution.



### What is REPL in Node.js?

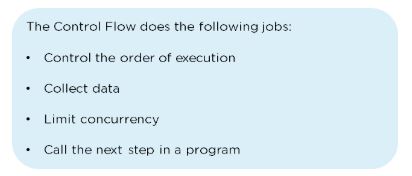
REPL stands for Read Eval Print Loop, and it represents a computer environment. It’s similar to a Windows console or Unix/Linux shell in which a command is entered. Then, the system responds with an output



### What is the control flow function?

The control flow function is a piece of code that runs in between several asynchronous function calls.

### How does control flow manage the function calls?



### What is the difference between fork() and spawn() methods in Node.js?

|  |  |
| --- | --- |
| **fork()** | **spawn()** |
| https://www.simplilearn.com/ice9/free_resources_article_thumb/fork.JPG | spawn |
| fork() is a particular case of spawn() that generates a new instance of a V engine. | Spawn() launches a new process with the available set of commands. |
| Multiple workers run on a single node code base for multiple tasks. | This method doesn’t generate a new V instance, and only a single copy of the node module is active on the processor. |

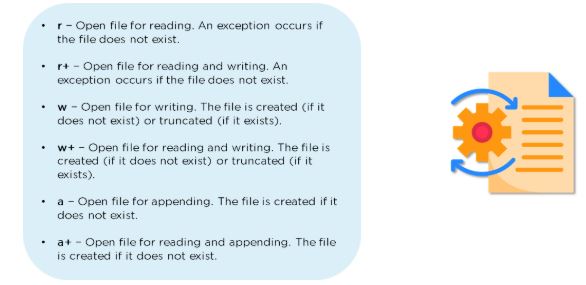
### What is the buffer class in Node.js?

The Buffer class in Node.js is used to handle binary data directly. It is a global class that provides a way of working with raw binary data, which is particularly useful when dealing with streams, files, and other I/O operations where data isn't always in a string format. Here's a breakdown of its features and usage:

### What is piping in Node.js?

Piping is a mechanism used to connect the output of one stream to another stream. It is normally used to retrieve data from one stream and pass output to another stream

### What are some of the flags used in the read/write operations in files?



### How do you open a file in Node.js?

openfile

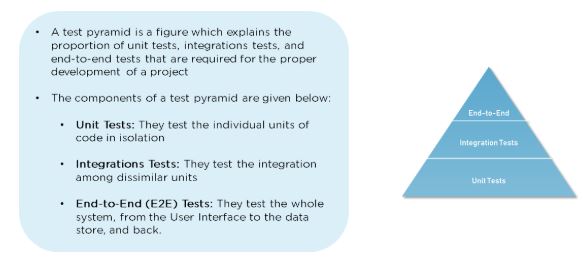
### 

### What is a reactor pattern in Node.js?

A reactor pattern is a concept of non-blocking I/O operations. This pattern provides a handler that is associated with each I/O operation. As soon as an I/O request is generated, it is then submitted to a demultiplexer

### 

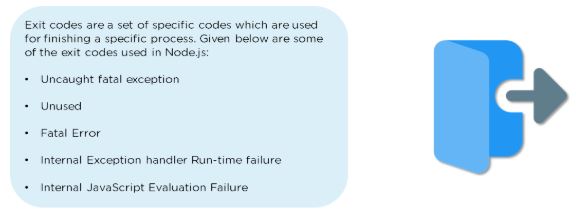
### What is a test pyramid in Node.js?



### For Node.js, why does Google use the V engine?

The V engine, developed by Google, is open-source and written in [C++](https://www.simplilearn.com/tutorials/cpp-tutorial/learn-cpp-basics). Google Chrome makes use of this engine. V, unlike the other engines, is also utilized for the popular Node.js runtime. V was initially intended to improve the speed of JavaScript execution within web browsers. Instead of employing an interpreter, V converts JavaScript code into more efficient machine code to increase performance. It turns JavaScript code into machine code during execution by utilizing a JIT (Just-In-Time) compiler, as do many current JavaScript engines such as SpiderMonkey or Rhino (Mozilla).

### Describe Node.js exit codes.



**What is middleware?**

Middleware in Node.js, particularly in the context of Express.js, refers to functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. These functions can execute code, make changes to the request and response objects, end the request-response cycle, and call the next middleware function.

**Key Points About Middleware:**

1. **Execution Order**: Middleware functions are executed sequentially in the order they are defined in the code.
2. **Types of Middleware**:

* **Application-level Middleware**: Bound to an instance of the app object using app.use() or app.METHOD().
* **Router-level Middleware**: Works in the same way as application-level middleware, except it is bound to an instance of express.Router().
* **Error-handling Middleware**: Defined with four arguments: err, req, res, and next. It handles errors that occur in the application.
* **Built-in Middleware**: Provided by Express, such as express.static, express.json, and express.urlencoded.
* **Third-party Middleware**: Created by the community, such as body-parser, cookie-parser, etc.

**Example of Middleware:**

const express = require('express');

const app = express();

// Application-level middleware

app.use((req, res, next) => {

console.log('Middleware function executed!');

next(); // Call the next middleware function

});

// Route handler

app.get('/', (req, res) => {

res.send('Hello World!');

});

// Error-handling middleware

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).send('Something broke!');

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

In this example:

* The first middleware logs a message for every request.
* The route handler sends a response for the root URL.
* The error-handling middleware handles any errors that occur in the application.

### What are the different types of HTTP requests?

**GET**

* **Purpose**: Retrieve data from a server.
* **Characteristics**:
  + Requests data from a specified resource.
  + Should not alter the server's state (idempotent).
  + Data is passed via the URL.
* **Example**:

GET /users HTTP/1.1

Host: example.com

**2. POST**

* **Purpose**: Submit data to a server to create/update a resource.
* **Characteristics**:
  + Sends data in the request body.
  + Can create a new resource or update an existing one.
  + Not idempotent (each request can result in a different outcome).
* **Example**:

POST /users HTTP/1.1

Host: example.com

Content-Type: application/json

{

"name": "John",

"email": "john@example.com"

}

**3. PUT**

* **Purpose**: Update a resource or create a new resource if it doesn't exist.
* **Characteristics**:
  + Sends data in the request body.
  + Idempotent (multiple identical requests should have the same effect).
* **Example**:

PUT /users/1 HTTP/1.1

Host: example.com

Content-Type: application/json

{

"name": "John",

"email": "john\_updated@example.com"

}

**4. DELETE**

* **Purpose**: Remove a specified resource.
* **Characteristics**:
  + Idempotent (multiple identical requests should have the same effect).
* **Example**:

DELETE /users/1 HTTP/1.1

Host: example.com

**5. PATCH**

* **Purpose**: Partially update a resource.
* **Characteristics**:
  + Sends data in the request body.
  + Not necessarily idempotent, though it can be.
* **Example**:

PATCH /users/1 HTTP/1.1

Host: example.com

Content-Type: application/json

{

"email": "john\_updated@example.com"

}

**6. HEAD**

* **Purpose**: Retrieve the headers of a resource, without the body.
* **Characteristics**:
  + Similar to GET, but without the response body.
  + Useful for checking what a GET request will return (e.g., to check for a resource's existence).
* **Example**:

HEAD /users HTTP/1.1

Host: example.com

**7. OPTIONS**

* **Purpose**: Describe the communication options for the target resource.
* **Characteristics**:
  + Can be used for CORS preflight requests.
  + Returns the HTTP methods supported by the server for the specified resource.
* **Example**:

OPTIONS /users HTTP/1.1

Host: example.com

**8. CONNECT**

* **Purpose**: Establish a tunnel to the server identified by the target resource.
* **Characteristics**:
  + Commonly used for SSL (HTTPS) connections through an HTTP proxy.
* **Example**:

CONNECT example.com:443 HTTP/1.1

Host: example.com

**9. TRACE**

* **Purpose**: Perform a message loop-back test along the path to the target resource.
* **Characteristics**:
  + Used for debugging purposes.
* **Example**:

TRACE /users HTTP/1.1

Host: example.com

### How would you connect a MongoDB database to Node.js?

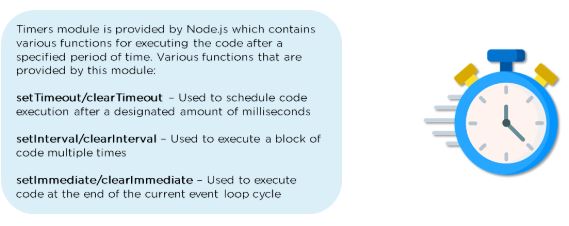
* Start by creating a MongoClient object
* Specify a connection URL with the correct IP address and the name of the database you want to create



### What is the purpose of NODE\_ENV?

### node-env

### List the various Node.js timing features.



### What is WASI, and why is it being introduced?

The WASI class implements the WASI system called API and extra convenience methods for interacting with WASI-based applications. Every WASI instance represents a unique sandbox environment. Each WASI instance must specify its command-line parameters, environment variables, and sandbox directory structure for security reasons.

### How do you manage packages in your Node.Js project?

Managing packages in your Node.js project is done using the Node Package Manager (NPM), which allows you to install and manage third-party packages and create and publish your packages.

### How is Node.js better than other frameworks?

Node.js is a server-side JavaScript runtime environment built on top of the V8 JavaScript engine, the same engine that powers Google Chrome. It makes Node.js very fast and efficient, as well as highly scalable.

### What is a fork in node JS?

The Fork method in Node.js creates a new child process that runs a separate Node.js instance and can be useful for running CPU-intensive tasks or creating a cluster of Node.js servers.

In Node.js, a fork refers to the creation of a new process that runs concurrently with the parent process. This is typically done using the child\_process module, which allows you to create child processes to execute code in parallel. Forking is especially useful for performing tasks that are CPU-intensive or for running separate instances of a Node.js application.

### What is the purpose of the module.exports?

The module. exports object in Node.js is used to export functions, objects, or values from a module and is returned as the value of the require() function when another module requires a module.

### What tools can be used to assure consistent code style?

In summary, several tools can be used in Node.js to ensure consistent code style and improve code quality, including ESLint, Prettier, and Jest.

### What is the difference between JavaScript and Node.js?

Node.js is a runtime environment for executing JavaScript code outside of a web browser, while JavaScript is a programming language that can be executed in both web browsers and Node.js environments.

### What is the difference between asynchronous and synchronous functions?

Synchronous functions block the execution of other code until they are complete, while asynchronous functions allow other code to continue executing while they are running, making them essential for writing scalable Node.js applications.

### What are the asynchronous tasks that should occur in an event loop?

Asynchronous tasks that should occur in an event loop in Node.js include I/O operations, timers, and callback functions. By performing these tasks asynchronously, Node.js can handle a large number of concurrent requests without blocking the event loop.

### What is the order of execution in control flow statements?

In Node.js, control flow statements are executed in a specific order. The order of execution is determined by the event loop. The event loop is a mechanism in Node.js that allows for the execution of non-blocking I/O operations.

### What are the input arguments for an asynchronous queue?

An asynchronous queue in Node.js is a data structure that allows for the execution of functions in a specific order. Functions are added to the queue and are executed in the order that they were added. An asynchronous queue is useful when you want to execute a series of functions in a specific order.

### Are there any disadvantages to using Node.js?

Node.Js is not suitable for CPU-intensive tasks. This is because Node.js is single-threaded, meaning it can only execute one task at a time. Node.js is not suitable for applications that require a lot of memory. This is because Node.js uses a lot of memory for each connection. If you have a large number of connections, it can quickly consume a lot of memory.

### What is the primary reason for using the event-based model in Node.js?

The main reason to use the event-based model in Node.js is performance. The event-based model allows for non-blocking I/O operations, which means that Node.js can handle a large number of connections without using a lot of resources.

### What is the difference between Node.js and Ajax?

Ajax and Node.js are two different technologies that are used for different purposes. Ajax is a client-side technology that allows for asynchronous communication between the client and the server. It is typically used to update parts of a web page without requiring a full page reload.

Node.js, on the other hand, is a server-side technology that is used for building fast, scalable, and efficient server-side applications. It is typically used for real-time applications, such as chat applications, online games, and streaming services.

### What is the advantage of using Node.js?

Node.js is fast and scalable. Node.js is easy to learn and use. Node.js is well-suited for real-time applications, such as chat applications, online games, and streaming services. This is because Node.js can handle a large number of connections and can perform non-blocking I/O operations, which makes it ideal for real-time communication.

### Can you access DOM in Node?

No, you cannot access the DOM in Node.js. The DOM is a browser-specific API that allows for the manipulation of HTML and XML documents. Since Node.js does not run in a browser, it does not have access to the DOM.

### Why is Node.JS quickly gaining attention from JAVA programmers?

Node.js is quickly gaining attention from Java programmers because it is fast, scalable, and efficient. Java is a popular server-side technology, but it can be slow and resource-intensive. Node.js, on the other hand, is built on the V8 JavaScript engine, which is known for its speed and performance.

### What are the Challenges with Node.js?

Node.js is single-threaded, which means that it can only execute one task at a time. Node.js is relatively new compared to other server-side technologies, such as Java and PHP. This means that there needs to be more support and more resources available for Node.js. Node.js is only suitable for applications that require a little memory.

### What is "non-blocking" in node.js?

In Node.js, non-blocking refers to the ability of the runtime environment to execute multiple tasks simultaneously without waiting for the completion of one task before starting the next. This is achieved through the use of asynchronous I/O operations, which allow Node.js to handle multiple requests concurrently.

### How does Node.js overcome the problem of blocking I/O operations?

Node.js uses an event-driven, non-blocking I/O model that allows it to handle I/O operations more efficiently. By using callbacks, Node.js can continue processing other tasks while waiting for I/O operations to complete. This means that Node.js can handle multiple requests simultaneously without causing any delays. Additionally, Node.js uses a single-threaded event loop architecture, which allows it to handle a high volume of requests without any issues.

### How can we use async await in node.js?

To use async/await in Node.js, you'll need to use functions that return promises. You can then use the async keyword to mark a function as asynchronous and the await keyword to wait for a promise to resolve before continuing with the rest of the code.

### Why should you separate the Express app and server?

Firstly, separating your app and server can make it easier to test your code. By separating the two, you can test your app logic independently of the server, which can make it easier to identify and fix bugs.

Secondly, separating your app and server can make it easier to scale your application. By separating the two, you can run multiple instances of your app on different servers, which can help to distribute the load and improve performance.

Finally, separating your app and server can make it easier to switch to a different server if necessary. By keeping your app logic separate from your server logic, you can switch to a different server without having to make any major changes to your code.

### Explain the concept of stub in Node.js.

In Node.js, a stub is a function that serves as a placeholder for a more complex function. Stubs are typically used in unit testing to replace a real function with a simplified version that returns a predetermined value. By using a stub, you can ensure that your unit tests are predictable and consistent.

### What is the framework that is used majorly in Node.js today?

There are many frameworks available for Node.js, but the two most popular ones are Express and Koa.

### What are the security implementations that are present in Node.js?

One of the most important security features in Node.js is the ability to run code in a restricted environment. This is achieved through the use of a sandboxed environment, which can help to prevent malicious code from accessing sensitive data or causing any damage to the system.

Another important security feature in Node.js is the ability to use TLS/SSL to encrypt data in transit. This can help to prevent eavesdropping and ensure that sensitive data is protected.

### What is Libuv?

Libuv is a critical component of Node.js, and it's what makes it possible to handle I/O operations in a non-blocking and efficient manner.

### What are global objects in Node.js?

Global objects in Node.js are objects that are available in all modules without the need for an explicit require statement. Some of the most commonly used global objects in Node.js include process, console, and buffer.

### Why is assert used in Node.js?

An assert module is an important tool for writing effective tests in Node.js.

### Why is ExpressJS used?

Express is a great choice for building web applications in Node.js, and its popularity and active community make it a safe and reliable choice for developers of all levels.

### What is the use of the connect module in Node.js?

The Connect module can be used to handle different types of middleware, such as error-handling middleware, cookie-parsing middleware, and session middleware. Error-handling middleware is used to handle errors that occur during the request/response cycle. Cookie parsing middleware is used to parse cookies from the request header. Session middleware is used to manage user sessions.

### What's the difference between 'front-end' and 'back-end' development?

Front-end developers focus on the client side of the application, while back-end developers focus on the server side of the application. Both roles are important for building a successful web application and require different skill sets and expertise.

### What are LTS releases of Node.js?

LTS stands for Long-term support. LTS releases of Node.js are versions that are supported for an extended period, usually for months from the time of release. These releases are typically more stable and reliable than non-LTS releases and are recommended for production use.

### What do you understand about ESLint?

ESLint is a popular open-source tool that is used to analyze and flag errors and potential problems in JavaScript code.

### Define the concept of the test pyramid. Please explain the process of implementing them in terms of HTTP APIs.

The test pyramid is a concept that is often used in software testing to illustrate the ideal distribution of different types of tests. The pyramid consists of three layers: unit tests, integration tests, and end-to-end tests. The idea is that the majority of tests should be at the unit level, with fewer tests at the integration and end-to-end levels.

To implement the test pyramid in terms of HTTP APIs, you can start by writing unit tests for each endpoint in the API. These tests should focus on testing the functionality of the endpoint in isolation without making any external requests or dependencies. Once the unit tests are passed, you can write integration tests that test the interaction between different endpoints and components in the API. Finally, you can write end-to-end tests that test the entire API, from the user interface to the database.

### How does Node.js handle the child threads?

Node.js handles child threads by creating separate instances of the Node.js runtime environment that can be used to execute code in parallel with the main process.

### What is an Event Emitter in Node.js?

An Event Emitter is a Node.js module that facilitates communication between objects in a Node.js application. It is an instance of the EventEmitter class, which provides a set of methods to listen for and emit events. In Node.js, events are a core part of the platform, and they are used to handle asynchronous operations.

### What is a thread pool, and which library handles it in Node.js?

A thread pool is a collection of threads that are used to execute tasks in parallel. In Node.js, the thread pool is handled by the libuv library, which is a multi-platform support library that provides asynchronous I/O operations.

### How are worker threads different from clusters?

Worker threads and clusters are two different approaches to leveraging the power of multiple CPUs in Node.js. While clusters create multiple instances of a Node.js process, each running on a separate CPU core, worker threads provide a way to create multiple threads within a single process.

### How to measure the duration of async operations?

The console.time and console.timeEnd methods allow you to measure the duration of a block of code. The console.time method is used to start the timer and the console.timeEnd method is used to stop the timer and log the duration to the console.

The performance.now method provides a more precise way to measure the duration of async operations. It returns the current timestamp in milliseconds, which can be used to calculate the duration of a task.

### How to measure the performance of async operations?

There are several tools and techniques you can use to measure performance, including using the built-in --prof flag, using the perf tool, and using third-party libraries like benchmark.js.

### What is meant by tracing in Node.js?

Tracing is a technique used in Node.js to profile the performance of an application. It involves recording the function calls and events that occur during the execution of the application and analyzing the data to identify performance bottlenecks.

### Where is package.json used in Node.js?

The package.json file is located in the root directory of an application and it is used by the npm package manager to install and manage the dependencies of an application.

### What is the difference between readFile and create Read Stream in Node.js?

Create Read Stream is a better option for reading large files, while the read file is a better option for small files. It is important to choose the right method based on the size of the file and the requirements of the application.

### What is the use of the crypto module in Node.js?

The crypto module is widely used in Node.js applications to generate secure random numbers, create digital signatures, and verify signatures. It also provides support for various encryption algorithms such as AES, DES, and RSA.

### What is a passport in Node.js?

Passport is a popular authentication middleware for Node.js. It provides a simple and modular way to implement authentication in Node.js applications. Passport supports many authentication mechanisms, including username/password, social logins like Facebook and Google, and JSON Web Tokens (JWTs).

### How to get information about a file in Node.js?

In Node.js, the fs module provides methods for working with the file system. To get information about a file, you can use the fs. stat() method. The fs. stat() method returns an object that contains information about the file, such as the file size, creation date, and modified date.

### How does the DNS lookup function work in Node.js?

In Node.js, the DNS module provides methods for performing DNS lookups. DNS stands for Domain Name System, and it is responsible for translating domain names into IP addresses. The DNS. lookup() method is used to perform a DNS lookup and resolve a domain name into an IP address.

### What is the difference between setImmediate() and setTimeout()?

The setTimeout() method schedules code execution after a specified delay, measured in milliseconds. On the other hand, the setImmediate() method schedules code execution to occur immediately after the current event loop iteration completes. This means that setImmediate() has a higher priority than setTimeout().

### Explain the concept of Punycode in Node.js.

Punycode is a character encoding scheme used in the domain name system (DNS) to represent Unicode characters with ASCII characters. It is used to encode domain names that contain non-ASCII characters, such as Chinese or Arabic characters.

### Does Node.js provide any Debugger?

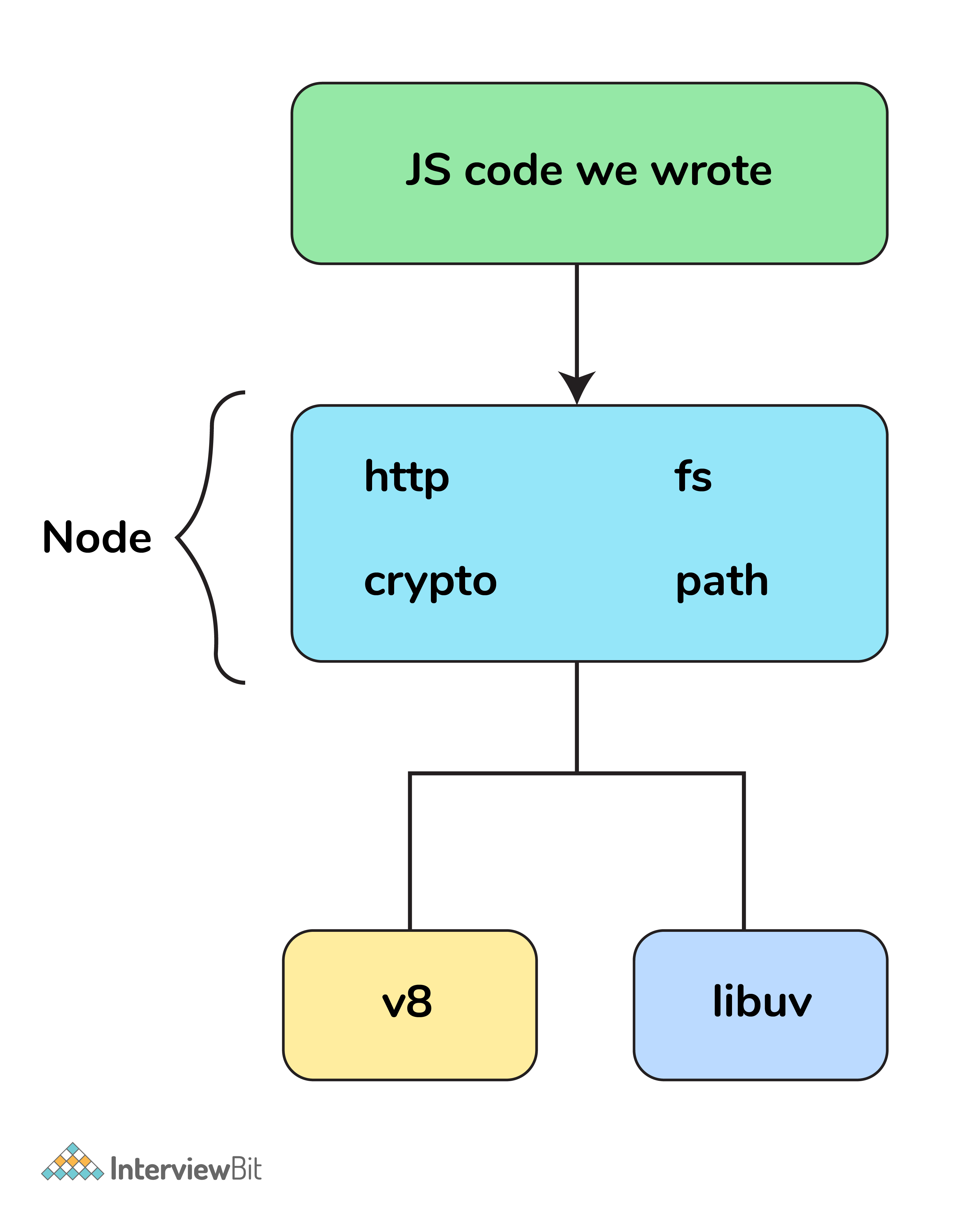
Yes, Node.js provides a built-in debugger that can be used to debug Node.js applications.

### For Node.js, why Google uses V8 engine?

Well, are there any other options available? Yes, of course, we have [Spidermonkey](https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey) from Firefox, Chakra from Edge but Google’s v8 is the most evolved(since it’s open-source so there’s a huge community helping in developing features and fixing bugs) and fastest(since it’s written in c++) we got till now as a JavaScript and WebAssembly engine. And it is portable to almost every machine known.

### What is a thread pool and which library handles it in Node.js

The Thread pool is handled by the libuv library. libuv is a multi-platform C library that provides support for asynchronous I/O-based operations such as file systems, networking, and concurrency.



### Why should you separate Express app and server?

The server is responsible for initializing the routes, middleware, and other application logic whereas the app has all the business logic which will be served by the routes initiated by the server. This ensures that the business logic is encapsulated and decoupled from the application logic which makes the project more readable and maintainable

### What is middleware?

Middleware comes in between your request and business logic. It is mainly used to capture logs and enable rate limit, routing, authentication, basically whatever that is not a part of business logic. There are third-party middleware also such as body-parser and you can write your own middleware for a specific use case.

### ****What is the difference between Synchronous and Asynchronous functions?****

| **Feature** | **Synchronous Functions** | **Asynchronous Functions** |
| --- | --- | --- |
| Execution Blocking | Blocks the execution until the task completes. | Does not block the execution; allows other tasks to proceed concurrently. |
| Waiting for Completion | Executes tasks sequentially; each task must complete before the next one starts. | Initiates tasks and proceeds with other operations while waiting for completion. |
| Return Value | Returns the result immediately after completion. | Typically returns a promise, callback, or uses event handling to handle the result upon completion. |
| Error Handling | Errors can be easily caught with try-catch blocks. | Error handling is more complex and often involves callbacks, promises, or async/await syntax. |
| Usage Scenario | Suitable for simple, sequential tasks with predictable execution flow. | Ideal for I/O-bound operations, network requests, and tasks requiring parallel processing. |