Node

>POST is used to submit data to be processed to a specified resource
.POST is not idempotent, meaning repeating a POST request may result in multiple resource creations or side effects.

>PUT is used to update or replace a resource at a specific URL.
.If the resource does not exist, PUT can also create it.
.PUT is idempotent, meaning repeating the same PUT request will not change the result after the initial application

>PREFLIGHT REQUEST
.A preflight request is a preliminary request made by the browser to check if the actual request is safe to send.
.This mechanism is part of the Cross-Origin Resource Sharing (CORS) protocol, which is used to manage cross-origin HTTP requests.

>EVENTEMITTER
.In Node.js, the EventEmitter class is part of the events module and is used to handle and manage events and listeners.
.It allows objects to emit events and allows other objects to listen and respond to those events.

>Authentication: Verifies the identity of a user or system. It ensures that the user is who they claim to be.
>Authorization: Determines what an authenticated user or system is allowed to do. It controls access to resources based on permissions

>Exit codes (or exit status codes) are numerical values returned by a process or script to indicate how it completed its execution.
.These codes are used to communicate the outcome of the process to the operating system or to other processes.
.They are particularly useful for error handling and debugging.

STREAM
.In Node.js, a stream is a fundamental concept for handling data in a continuous and efficient manner.
.Instead of loading entire datasets into memory at once, streams process data in chunks, making them ideal for large files or real-time data processing.
.Streams are continuous flows of data that are processed in chunks.
.**Readable Streams**: Used for reading data
.**Writable Streams**: Used for writing data.
.**Duplex Streams**: Streams that are both readable and writable
.**Transform Streams**: A type of duplex stream that can modify the data as it is read or written

>A buffer is a raw binary data storage object that represents a chunk of memory.
.It is used to work with binary data directly.

>HTTP ONLYCOOKIES
.HTTP-only cookies are a type of cookie that is designed to be more secure by restricting access to the cookie's data from client-side scripts.
.They are used primarily to help prevent cross-site scripting (XSS) attacks from accessing sensitive information stored in cookies.

>BUILT IN MIDDLEWARE
.express.json():parses incoming json request bodies
.express.urlencoded:parses url encoded request bodes
.express.static:servers static file
.express.Router()creates modular router

>ROUTER CHAINING
.Route chaining in Express.js allows you to define multiple HTTP methods for the same route using a clean and organized syntax.
.This is useful for structuring your routes and keeping related logic together, making your code more readable and maintainable.

>Dynamic routing allows you to create routes that can handle variable paths and parameters.

>ROUTE/PATH PARAMETERS
.Route/path parameters are dynamic segments in the URL path that are used to capture values.
.They are part of the URL path and are defined using a colon (:) in the route pattern.
.Use when you want to identify a resource

>USE CASES
>RESOURCE IDENTIFICATION:
.Example: /users/123 where 123 is a path parameter representing the ID of a specific user. This is used to retrieve, update, or delete the user with that ID.

>HIERARCHIAL DATA:
Example: /departments/finance/employees where finance is a department and employees is a collection within that department.
Use Case: When you have a hierarchy or nested resources and want to access data at different levels of that hierarchy.

>QUERY PARAMETERS
.Query parameters are key-value pairs appended to the end of a URL after a question mark (?).
.They are used to pass additional data to the server in a more flexible way compared to route parameters.
.Use for filtering ,pagination ...
>USE CASES

>FILTERING DATA
Example: /products?category=electronics&price_max=500 where category and price_max are query parameters used to filter products.
Use Case: When you need to apply filters to a collection of resources or data. For example, searching for items within a specific category or price range.

>SORTING DATA
Example: /articles?sort=date where sort is a query parameter specifying how to sort the results.
Use Case: When you want to specify the order in which results should be returned, such as sorting by date, relevance, or other criteria.

>PAGINATION
Example: /users?page=2&limit=10 where page and limit are query parameters used for paginating results.
Use Case: When you need to control the pagination of results, specifying which page of results to return and how many items per page.

>SEARCH QUERIES:
Example: /search?q=keyword where q is a query parameter representing the search term.
Use Case: When performing searches or queries where the input can vary and affects the results returned.

CLUSTER
.The cluster module in Node.js is designed to create child processes (workers) that share the same server port.
.This allows you to run multiple instances of the node.js application simultaneously, each on a different core of the CPU.

WORKER
.A worker is a child process that is spawned by the cluster master process.
.Each worker runs its own instance of the Node.js event loop, and they can share the same server port.
.Workers are essentially identical copies of the main process, except they don't automatically inherit file descriptors from the parent.

THREADPOOL
.A thread pool is a design pattern used in concurrent programming to manage a pool of worker threads that can be reused to perform multiple tasks.
.Instead of creating and destroying a new thread for every single task, a thread pool maintains a fixed number of threads that are ready to execute tasks.
.When a task is submitted, it's assigned to an available thread from the pool
.If all threads are busy, the task waits in a queue until a thread becomes available.

>EVENT-DRIVEN PROGRAMMING
.Event-driven programming is a programming paradigm where the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs or threads.
.In this model, an event loop continuously listens for events and triggers the corresponding event handler (callback function) when an event is detected.

SPAWN
.The spawn method is used to launch a new process to execute a command or script.
.It can be used to run a shell command, starting another Node.js script, or executing any executable file.

FORK
.The fork method is a special case of spawn that is specifically designed to spawn new Node.js processes.
.It is used to create a new instance of the V8 engine and execute a Node.js module.
.Unlike spawn, fork automatically establishes a communication channel (IPC) between the parent and child processes, making it easier to send messages between them.

EXEC
.In Node.js, the exec method is function provided by the child_process module.
.It is used to execute shell commands and buffers the output of the command in memory, returning it as a single string when the command finishes executing.

EXECFILE
.execFile method in the child_process module, which is similar to exec but designed specifically for executing files directly without a shell

---

APP.LOCALS
.app.locals is used to store local variables that are accessible throughout the application.

RATE LIMITING
.Rate limiting is a technique used to control the amount of incoming or outgoing traffic to or from a network, service, or API.
.The primary goal of rate limiting is to prevent abuse, ensure fair usage, and maintain the performance and stability of a system by limiting the number of requests a user or client can make within a given time period.

CONTENT NEGOTIATION
.Content negotiation is used to determine and deliver the most appropriate representation of a resource based on the client's capabilities and preferences.
.It is a crucial aspect of HTTP-based communication, particularly in RESTful APIs, where clients may

request different formats of data.
.Resources on a server can have multiple representations, such as JSON, XML, HTML, etc.
.Content negotiation allows the server to choose the most suitable representation based on the client’s request.
.Done through accept in the header

OPTIONS
.The OPTIONS method in HTTP is used to describe the communication options for a given resource or server.
.It is part of the HTTP specification and is primarily used for discovering the capabilities of a server or resource, or for implementing cross-origin resource sharing (CORS) in web applications.

LOCAL STORAGE
.Part of the Web Storage API
.Data in localStorage persists even after the browser is closed and reopened.
.It remains until it is explicitly deleted by the user
.Data stored in localStorage is shared across all tabs and windows

SESSION STORAGE
.Part of the Web Storage API
.Data in sessionStorage only lasts for the duration of the page session
.This means that the data is cleared when the page session ends, such as when the tab or browser window is closed.
.Data stored in sessionStorage is scoped to the tab or window that created it
.Each tab or window has its own separate sessionStorage, even if they are from the same origin.
.sessionStorage is useful for storing temporary data that only needs to be available during the current session, like form inputs, temporary calculations, or state that doesn't need to persist beyond the tab/window.

>UTIL MODULE
.The util module in Node.js is a built-in module that provides a collection of utility functions that can be used for various tasks such as debugging, formatting, and working with asynchronous code.

>UTIL.FORMAT()
.Formats a string using placeholders

>UTIL.PROMSIFY()
.Converts a callback-based function to a function that returns a promise.

>UTIL.CALLBACKIFY()
.Converts an async function (that returns a promise) into a callback-based function.

>UTIL.INHERITS()
.Sets up prototype inheritance between two constructors

>PROCESS
.A process is an instance of a running program.
.It has its own memory space, operating system resources, and execution context
.Each process operates independently, preventing interference between different programs.
.Processes consume significant system resources, making them less efficient for tasks that require multiple concurrent operations.

>THREADS
. A thread is a lightweight unit of execution within a process
. It shares the process's memory space and resources but has its own execution stack.
.Multiple threads can run concurrently within a single process, enabling efficient handling of tasks that can be divided into smaller, independent subtasks.

>ERROR-FIRST-CALLBACK
.The error-first callback pattern is typically used in functions that perform asynchronous operations.
.The pattern involves passing a callback function as an argument to an asynchronous function.
.This callback function is called with two arguments:
.Error Argument: The first argument is an error object.
.If an error occurs during the operation, this argument contains information about the error.
.Result Argument: The second argument contains the result of the asynchronous operation (if successful).

>ACCESS TOKEN
.An access token is a short-lived token that is issued by an authentication server after a user successfully logs in.
.It is used to authenticate and authorize requests to a server (e.g., API requests).
.The token typically includes information about the user, permissions, and an expiration time.
.Access tokens have a short lifespan (e.g., 5 minutes to 1 hour) to minimize the risk of misuse if the token is stolen.
.Access tokens are designed to expire quickly, requiring re-authentication or the use of a refresh token.

>REFRESH TOKEN
.A refresh token is a long-lived token that is also issued by the authentication server alongside the access token.
.It is used to obtain a new access token without requiring the user to re-authenticate.

>PROCESS.NEXTTICK
.process.nextTick() is a special function in Node.js that allows you to schedule a callback function to be

invoked in the next iteration of the event loop, before any I/O operations or timers.
.It’s a way to defer the execution of a function until the current operation is complete, but before any I/O tasks or timers in the event loop.
.This gives it higher priority than other callbacks, like those scheduled with setTimeout() or setImmediate().
.It is placed in the microtask queue

>EXPRESS.URLENCODED
.express.urlencoded is a middleware function provided by the Express.js framework for handling URL-encoded form data.
.When you use this middleware, Express.js can parse incoming requests with URL-encoded payloads (typically from HTML forms) and populate req.body with the parsed data.

>CSRF
.Cross-Site Request Forgery (CSRF) is a type of security vulnerability in web applications where an attacker tricks a user into performing actions on a web application where they are authenticated.
.CSRF exploits the trust that a website has in a user's browser.
.Prevent CSRF Cookies
1. Anti-CSRF Tokens
2. SameSite Cookie Attribute
3. Custom Request Headers

>LIBUV
.libuv is a multi-platform C library that provides support for asynchronous I/O operations.
.It serves as the backbone for Node.js's non-blocking I/O model, enabling it to handle multiple tasks concurrently without blocking the main event loop.
.libuv implements the core event loop mechanism in Node.js.
.This is the central part of Node.js's asynchronous and non-blocking architecture.
.The event loop listens for events (like I/O completion, timers, etc.) and dispatches callbacks accordingly.
.libuv provides non-blocking I/O operations, such as reading from or writing to files, making network requests, handling DNS queries, and more.
.These operations are performed in the background without stopping the execution of other code.
.Thread Pool: For some operations that can’t be performed asynchronously (like file system operations or some DNS operations), libuv uses a thread pool.
.This pool of worker threads handles blocking tasks in the background, freeing the event loop to handle other tasks.

>V8 ENGINE
.The V8 engine is an open-source, high-performance JavaScript engine developed by Google.
.It is written in C++ and is used in both Google Chrome and Node.js to run JavaScript code.
.JavaScript Execution: V8 compiles and executes JavaScript code.
.It translates JavaScript into machine code that can be executed directly by the system's CPU, making it extremely fast.

.Just-In-Time (JIT) Compilation: V8 uses JIT compilation to improve performance.
.Instead of interpreting JavaScript line by line, V8 compiles the entire JavaScript code into machine code before executing it, making execution faster.
.Memory Management: V8 has an efficient memory management system that includes garbage collection.
.It automatically allocates and deallocates memory used by JavaScript objects and variables, freeing up memory when it's no longer needed.

>COMMONJS
.CommonJS is the original module system used in Node.js.
.It uses the require() function to import modules and module.exports to export them.
.CommonJS modules are loaded synchronously, which is fine for server-side environments where modules are typically loaded once at startup.
.Once a module is loaded, it is cached. Subsequent require() calls for the same module return the cached version.

>ES6 MODULES
.ES6 Modules are a standardized module system introduced in ECMAScript 2015 (ES6).
.Node.js has supported ES6 modules natively since version 12, allowing developers to use import and export syntax.
.Asynchronous Loading: ES6 modules can be loaded asynchronously using dynamic imports.
.Improved Syntax: Uses import and export keywords, making it more readable and concise.

>LOCAL MODULES
.Local modules are modules that are specific to a project and are installed in the project’s directory (usually in the node_modules folder).
.They are only accessible within that project.
.Local modules are meant to be used only within the project they are installed in.

>GLOBAL MODULES
.Global modules are installed at the system level and can be accessed from any project on the machine.
.They are typically used for command-line tools or utilities that need to be available system-wide.

>MODULE CACHING
.Module caching enhances performance by avoiding the reloading of modules that have already been required.
.When a module is loaded for the first time, Node.js caches it, so subsequent calls to require that module return the cached version instead of loading it again.
.This mechanism helps improve application efficiency and reduces the overhead of repeatedly loading the same module.

>EXPRESS
.Express.js is a web application framework for Node.js designed to simplify the process of building web

applications and APIs.
.It provides a robust set of features to develop both web and mobile applications quickly and efficiently.
.It provides features like routing, middleware support, and template engines, making it easier to handle HTTP requests and responses.

>MIDDLEWARE
.Middleware in Express.js refers to functions that have access to the request (req), response (res), and the next middleware function in the application's request-response cycle.
.Middleware functions can perform various tasks, such as executing code, modifying request and response objects, ending the request-response cycle, and calling the next middleware function.
>Types of Middleware
.Application-Level Middleware: Defined at the application level using app.use() or route-specific methods like app.get(), app.post(), etc.
.Router-Level Middleware: Similar to application-level middleware but is applied to specific routers created using express.Router().
.Built-in Middleware: Express provides some built-in middleware like express.json() for parsing JSON bodies and express.static() for serving static files.
.Third-Party Middleware: Middleware developed by the community that can be installed via npm (e.g., body-parser, morgan, helmet).

.Application-Level Middleware is applied globally to all routes in an Express application, while Router-Level Middleware is applied specifically to routes defined within a particular router.

>ERROR HANDLING MIDDLEWARE
.Error-handling middleware in Express.js is a special type of middleware designed to catch and process errors that occur during the request-response cycle.
.It allows you to define centralized error handling logic, making your application more robust and easier to maintain.

>ROUTING
.Routing in Express.js refers to the mechanism that allows you to define how an application responds to client requests for specific endpoints (URLs) based on the HTTP method (GET, POST, PUT, DELETE, etc.).
.It is an essential part of building web applications and APIs, enabling you to manage different paths and their corresponding functionalities.

>REQ OBJECT
.The req object contains information about the HTTP request made by the client.
.req.params: An object containing route parameters defined in the URL. For example, in the route /user/:id, req.params.id would give you the value of the id.
.req.query: An object containing the query string parameters. For example, in the URL /search?term=apple, req.query.term would be apple.
.req.body: An object containing the parsed body of the request. This is populated when you use middleware like express.json() or express.urlencoded(). It contains data sent in the body of POST or

PUT requests.
.req.method: The HTTP method used in the request (e.g., GET, POST).
.req.url: The full URL of the request.

>RES OBJECT
.The res object is used to send back a response to the client.
.res.send(): Sends a response of various types (string, object, buffer, etc.). Express automatically sets the correct Content-Type header based on the data type.
.res.json(): Sends a JSON response and sets the Content-Type to application/json.
.res.status(): Sets the HTTP status code for the response. This method can be chained with other response methods (e.g., res.status(404).send('Not Found')).
.res.redirect(): Redirects the client to a different URL.
.res.render(): Renders a view template (used in server-side rendering scenarios).

>TEMPLATE ENGINE
.A template engine is a tool used in web development to dynamically generate HTML by combining static templates with data provided by the server.
.In web applications, a template engine simplifies the process of rendering HTML pages on the server-side by embedding dynamic data (e.g., from a database) into HTML templates.

>SESSION
.Session management allowing us to store user-specific data across multiple requests.
.In Express.js, session management is typically handled using middleware.
.The most common approach is to use the express-session middleware, which facilitates the creation, management, and storage of user sessions.
.Key Options in express-session
.secret: A secret key used to sign the session ID cookie. Keep it safe and secure.
.resave: When set to false, it prevents the session from being saved back to the store if it wasn’t modified during the request.
.saveUninitialized: When set to true, it forces uninitialized sessions to be saved to the store. This can be useful for tracking sessions even if they aren’t modified.
.cookie: An object to set cookie options such as secure (true for HTTPS), maxAge (duration for which the session is valid), etc..

>CACHE
.cache is a temporary storage mechanism used to store frequently accessed data for faster retrieval.
.Instead of fetching data from a slower source repeatedly, caching allows you to store the results of expensive operations in memory (RAM) so that future requests can access the cached data much faster.

>NEXT()
.next is a function that is used to pass control to the next middleware function in the request-response cycle.
.If a middleware function does not end the cycle (e.g., by sending a response), it needs to pass control

to the next middleware, and this is done by calling next().
.If any middleware function does not call next(), the request would not continue to the next function, and the response wouldn't be sent.

>SOCKETS
.Sockets (specifically WebSockets) provide a way for real-time, bidirectional communication between a client and a server.
.While HTTP follows a request-response model, WebSockets allow the server to push data to the client without the client having to request it.

>HOW NODEJS HANDLES CONCURRENCY
1.EVENT LOOP
.Node.js has a single-threaded event loop that handles multiple I/O operations (like reading files, making HTTP requests, or accessing databases) asynchronously.
.Instead of waiting for each operation to complete, the event loop can handle other tasks in the meantime.

2. ASYNCHRONOUSE ,NON-BLOCKING I/0
.When you make a request (e.g., file read, HTTP request) in Node.js, it is usually handled in a non-blocking way.
.This means the request is initiated, but Node.js doesn't wait for it to finish before moving on to the next task.
.When the operation is done, a callback function (or a Promise/async-await) is triggered, notifying the event loop that the operation has finished and the result can be processed.

3. LIBUV AND THREAD POOL
.Node.js internally uses a library called libuv.
.Libuv creates a thread pool (usually 4 threads by default) for handling more complex I/O tasks (like file system operations, DNS resolution, or cryptography).
.These tasks are sent to the thread pool, which works concurrently in the background, and once they are completed, the results are returned to the event loop for further processing.

>CRON JOB
.A cron job in Node.js is a task or script that runs automatically at scheduled intervals, much like tasks scheduled in the Linux cron system.
.In Node.js, you can implement cron jobs using the node-cron package, which allows you to schedule tasks to run periodically (e.g., every minute, hour, or day) based on the standard cron syntax.

>COOKIE
.Cookies are small pieces of data that are stored on the user's computer by the web browser while browsing a website.
.They are used for various purposes and play a crucial role in how web applications function.
.Disadvantage of cookie
1.Privacy Concerns

2.User data stored in cookies can be shared with third parties, increasing the risk of personal information being misused.
3.Security Risks
4.Storage Limitationsx

>HTTP HEADERS
.HTTP headers are key-value pairs that are sent between the client and the server in an HTTP request or response.
.They provide additional information about the request or response, or about the client or server.
There are two main types of HTTP headers:
1.Request headers: Sent by the client to provide information about the request.
2.Response headers: Sent by the server to provide information about the response.
.It includes the type of the data that the user wants,Authorization,Content-Type...

>FS.STAT
.The fs.stat method is part of the Node.js fs (filesystem) module, which provides an API to interact with the file system.
.The fs.stat method is used to retrieve information about a file or directory.
.It returns a fs.Stats object containing various details about the specified file or directory, such as its size, creation time, modification time, and permissions.

>PARTS OF HTTP REQUEST
1.REQUEST LINE
.Method: The type of request being made (e.g., GET, POST, PUT, DELETE, etc.).
.Request URI: The resource being requested (e.g., /index.html).
.HTTP Version: The version of the HTTP protocol being used (e.g., HTTP/1.1).

2.HEADERS -additional information
.Host: Specifies the domain name of the server (e.g., Host: www.example.com).
.User-Agent: Information about the client (browser) making the request (e.g., User-Agent: Mozilla/5.0).
.Accept: Indicates the content types that the client can process (e.g., Accept: text/html).
.Content-Type: Indicates the media type of the resource being sent (especially in POST requests) (e.g., Content-Type: application/json).

3.BODY(optional)

>PARTS OF HTTP RESPONSE
1.STATUS LINE
.HTTP Version: The version of the HTTP protocol used.
.Status Code: A three-digit code indicating the result of the request (e.g., 200, 404, 500).
.Status Message: A brief description of the status code (e.g., OK, Not Found, Internal Server Error).

2.HEADERS -additional information
.Content-Type: The media type of the response body (e.g., Content-Type: application/json).

.Content-Length: The length of the response body in bytes (e.g., Content-Length: 123).
.Set-Cookie: Used to send cookies from the server to the client (e.g., Set-Cookie: sessionId=abc123; Path=/).

3.BODY(optional)

>ENV
.Environment variables are key-value pairs used to configure the runtime environment of a system or application.
.environment variables are often used to store sensitive data like API keys, database credentials, or configuration options

>IDEMPOTENCY
.Idempotency refers to the property of an operation where performing it multiple times produces the same result as performing it once.

.Idempotent HTTP Methods
1.GET: Fetching a resource. Calling it multiple times will return the same resource without side effects.
2.PUT: Updating a resource. Sending the same update multiple times will result in the resource having the same state as after the first update.
3.DELETE: Removing a resource. Calling it multiple times will have the same effect as calling it once

.Non-idempotent HTTP Methods
POST: Creating a new resource. Each call can create a new resource, leading to different states.

>PUT
.PUT is used to update an existing resource or create a new resource if it does not exist.
.It typically replaces the entire resource with the new representation provided in the request.

>PATCH
.PATCH is used to apply partial modifications to a resource. Instead of sending the complete representation, you send only the fields that need to be updated

>CONCURRENCY VS PARALLELISM
1.Concurrency
.Concurrency is the ability to handle multiple tasks at the same time by interleaving their execution.
.It’s more about dealing with multiple things in progress at the same time, rather than literally running them simultaneously
.JavaScript achieves concurrency using asynchronous programming.
.This is possible because JavaScript is single-threaded but can manage asynchronous tasks using techniques like callbacks, promises, and async/await.
.Concurrency in JavaScript is achieved with asynchronous programming techniques, allowing tasks to be managed and handled efficiently without blocking the main thread.

2.Parallelism
.Parallelism is about executing multiple tasks at the same time using multiple cores or threads.
.This can be achieved by truly dividing the work across multiple workers
.JavaScript by nature (due to its single-threaded event loop) does not inherently have parallelism.
.However, you can achieve parallelism in JavaScript using Web Workers or Node.js's Worker Threads.
.Parallelism requires actual execution on separate threads or processes and is achieved using Web Workers in the browser or Worker Threads in Node.js.

>PROCESS MODULE
.The process module is a global module in Node.js that provides information and control over the current Node.js process.
.It is available globally without needing to require it using require().
.This module is useful for handling the runtime environment, accessing system information, and controlling the Node.js process.

>OS MODULE
.The os module provides utility functions and properties to access information about the operating system.
.Unlike the process module, which deals with the current Node.js runtime, the os module provides more generalized information about the underlying operating system.

>APP.USE
.app.use is used to register middleware in your Express application.

>APP.SET
.app.set is used to assign or configure a setting for the Express application.
.It is primarily used to set configurations or global values that can be accessed throughout your app.
.It modifies or defines application-level settings such as port numbers, view engines, directory paths, or custom application configurations

>RES.END
.Ends the response process.
.This method is typically used when you want to send a response without any data or when you've manually managed the response (like in streaming)

>STATUS CODES
1. Success Codes
.200 OK: The request was successful.
.201 Created: The request was successful, and a resource was created.
.204 No Content: The request was successful, but there is no content to send back.

2. Client Error Codes
.400 Bad Request: The request could not be understood by the server due to malformed syntax.
.401 Unauthorized: The request requires user authentication.

.403 Forbidden: The server understood the request, but it refuses to authorize it.
.404 Not Found: The requested resource could not be found.

3. Server Error Codes
.500 Internal Server Error: A generic error message indicating that the server encountered an unexpected condition.
.502 Bad Gateway: The server received an invalid response from the upstream server.
.503 Service Unavailable: The server is currently unable to handle the request due to temporary overload or maintenance.