- **CommonJS:-**
  CommonJS is a module system used in Node.js for organizing and structuring JavaScript code. It provides a way to split code into separate files (modules) and then import and export functionality between these modules.
  (1) - // math.js
  function add(a, b) {
  return a + b;
  }

  module.exports = { add };

  (2) –
  // app.js
  const { add } = require('./math');

  console.log(add(3, 4)); // Output: 7

- fs.stat:-
  In Node.js, the fs.stat method is used to retrieve information about a file or directory. It returns an instance of fs.Stats, which contains information such as file size, modification time, and whether the path points to a file or directory.

  ```
  const fs = require('fs');

  const path = '/path/to/file';

  fs.stat(path, (err, stats) => {
   if (err) {
    console.error('Error getting file stats:', err);
    return;
   }

   console.log('Last modified:', stats.mtime); // Output: Last modified: 2022-05-24T08:00:00.000Z

   // You can format the mtime if needed
   console.log('Formatted last modified:', stats.mtime.toLocaleString()); // Output: Formatted last modified: 5/24/2022, 1:00:00 PM
  });
  ```

- **promisify:-**
  promisify is a utility function in Node.js that is used to convert callback-based functions into Promise-based functions. It allows you to work with asynchronous APIs using Promises, which can simplify your code and make it more readable, especially when dealing with asynchronous operations.

  const fs = require('fs');

```
const util = require('util');

// Use promisify to convert fs.readFile from callback-based to Promise-based
const readFilePromise = util.promisify(fs.readFile);

// Usage of readFilePromise
readFilePromise('example.txt', 'utf8')
 .then(data => {
  console.log('File contents:', data);
 })
 .catch(error => {
  console.error('Error reading file:', error);
 });
```

- **Concurrency:-**
  Concurrency in Node.js refers to the ability of Node.js to handle multiple tasks simultaneously, allowing it to execute non-blocking I/O operations efficiently. Node.js is inherently designed to be non-blocking and asynchronous, which means that it can handle multiple operations concurrently without waiting for each operation to complete before moving on to the next one.

- **process.nextTick vs setImmediate:-**
  process.nextTick():-
  ***it queues a callback function to be executed on the next iteration of the event loop, immediately after the current operation completes and before any I/O events are fired.**
  *** Callbacks scheduled with process.nextTick() are executed before any I/O events, timers, or setImmediate callbacks. It's often used to defer execution of a callback until the current operation completes, allowing for more predictable and immediate execution of the callback.**
  ***It's not suitable for CPU-intensive tasks or recursive operations, as it can cause the event loop to block.**

=> process.nextTick() is used to defer execution of a callback until the current operation completes, while setImmediate() is used to execute a callback asynchronously, but after any I/O events that are already in the event loop queue. Both functions have their use cases, and choosing the right one depends on the specific requirements of your application.

- **Microtask:-**
  microtasks refer to tasks that are scheduled to be executed at the end of the current event loop iteration, after the current operation completes, but before the event loop moves on to the next iteration. Microtasks are typically used for scheduling small, high-priority tasks that need to be executed as soon as possible.

- **protection against JWT forgery:-**

  Protection against JWT forgery involves implementing measures to ensure the integrity and authenticity of JSON Web Tokens (JWTs) to prevent unauthorized tampering or modification by attackers.

  ***Use Strong HMAC Algorithms: When signing JWTs, use strong HMAC (Hash-based Message Authentication Code) algorithms**
  ***Protect Secret Keys: Keep the secret keys used for signing JWTs confidential and secure. Store them securely on the server-side and avoid exposing them in client-side code or configuration files.**

---

base64:-
Base64 is a binary-to-text encoding scheme that represents binary data in an ASCII string format by converting it into a set of 64 characters. It's commonly used for encoding binary data such as images, audio files, and cryptographic keys into a text format that can be safely transmitted over text-based protocols or stored in text-based formats like JSON or XML.

- **HTTP OPTIONS:-**

The HTTP OPTIONS method is used to request information about the communication options available for a given resource or server. It's a part of the HTTP protocol and is often used for cross-origin resource sharing (CORS) and to determine the supported methods and headers for a particular endpoint

---

- CORS?
  CORS (Cross-Origin Resource Sharing) is an HTTP header-based mechanism that allows the server to specify any other source from which the browser should obtain resources or send data. These sources can differ from the current by the hostname, HTTP scheme, or port number.

- CSRF:-
  CSRF (Cross-Site Request Forgery) is a type of security vulnerability that occurs when a malicious website tricks a user's browser into making unintended requests to a different website where the user is authenticated.

  In simpler terms, imagine you're logged into your bank's website in one browser tab, and you visit a malicious website in another tab. If the malicious website contains hidden code that instructs your browser to send requests to your bank's website (such as transferring money or changing your password), your browser will automatically send those requests because it's still logged into your bank. This happens without your knowledge or consent, and it can lead to unauthorized actions being taken on your behalf.

- **express.json vs express.urlencoded:**

1. `express.json()`:

- Think of JSON as a format for exchanging data between a web server and a client (like a browser or mobile app). It's structured like a JavaScript object.
- `express.json()` is a middleware in Express.js (a web framework for Node.js) that parses incoming requests with JSON payloads. In simpler terms, it helps your server understand JSON data sent by clients.
- When a client sends data to your server using JSON format (e.g., submitting a form with JSON data), `express.json()` parses that JSON data into a JavaScript object that your server can work with.
- Example: If a client sends `{ "name": "John", "age": 30 }` to your server, `express.json()` converts it into `{ name: 'John', age: 30 }`.

2. `express.urlencoded()`:

- Unlike JSON, URL-encoded data is sent as key-value pairs in the URL of an HTTP request. It's commonly used when submitting HTML forms.
- `express.urlencoded()` is another middleware in Express.js that parses incoming requests with URL-encoded payloads. It helps your server understand data sent in this format.
- When a client submits a form with data using the `application/x-www-form-urlencoded` content type, `express.urlencoded()` parses that data into a JavaScript object that your server can work with.
- Example: If a client submits a form with `name=John&age=30`, `express.urlencoded()` converts it into `{ name: 'John', age: '30' }`.

- **reverse proxying:-**

Reverse proxying is a technique used in web server configuration to distribute incoming requests to multiple backend servers or services based on certain criteria, such as the request URL, headers, or other conditions. In simpler terms, it's like having a middleman server that receives requests from clients and forwards them to the appropriate backend server to handle the request.

Here's how reverse proxying works:

1. **Client sends a request**: A client (e.g., a web browser) sends a request to a reverse proxy server.

2. **Reverse proxy receives the request**: The reverse proxy server receives the incoming request from the client.

- **Subdomain:-**
  A subdomain is a part of a larger domain name in the Domain Name System (DNS) hierarchy. It's used to organize and structure websites into distinct sections or categories. In simpler terms, you can think of a subdomain as a subset of a main domain that represents a specific area or purpose within that domain.

  Usage: Subdomains are commonly used to create separate sections or websites within a larger domain. For example: blog.example.com: A subdomain for hosting a blog.

- **load balancing:**

Load balancing is a technique used in computer networking and web server configuration to distribute incoming network traffic across multiple servers or resources. The goal of load balancing is to optimize resource utilization, maximize throughput, minimize response time, and ensure high availability and reliability of services. In simpler terms, it's like having multiple cashiers at a supermarket checkout to serve customers faster and more efficiently.

Here's how load balancing works:

1. **Incoming Requests**: When a client (such as a web browser) sends a request to access a website or service, it is received by a load balancer.

2. **Load Balancer**: The load balancer is a specialized server or hardware device that sits between the clients and the backend servers. Its job is to evenly distribute incoming requests across multiple backend servers based on predefined rules or algorithms.

- **Sudo:-**
  sudo is a command-line utility in Unix-like operating systems (such as Linux and macOS) that allows users to execute commands with elevated privileges. It stands for "superuser do".

  It allows users to perform these tasks without needing to log in as the root user, which reduces the risk of accidental damage to the system and enhances security by limiting access to privileged operations.

1. **Authorization**: When a user attempts to run a command with `sudo`, they are prompted to enter their own password to confirm their identity.

2. **Permission Check**: After entering the password, `sudo` checks whether the user is authorized to execute the specified command with elevated privileges.
