Advance Node.JS CheatSheet :

14 Middleware :

- Middleware functions are functions that have access to the req object (req) the response object (res) and the next middleware function in the application's request-response cycle.
- Commonly used in Express.js to handle tasks, such as logging, authentication and error handling

Ex:
```
Const express= require ('express');
Const app= express();

app.use ((req, res, next) => {   // Basic Middleware.
    Console.log ('Time :', Date.now());
    next();
});
```

15 Asynchronous programming :

- Promises represents the eventual Completion (or failure) of an asynchronous operation.
- Async/Await provides a way to work with promises in a more readable, synchronous like manner.

Ex:
```
Const fetchData = async () => {
    try {
        Const res = await fetch ("https://api.example.com/data");
        Const data= res.json();
    }
    Catch (error) {
        clg(error);
    };
fetchData();
```

3) Streams :

- Streams handle reading/writing data piece by piece, useful for large data.

Types: Readable, Writable, Duplex, Transform

Example :

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

    Const readableStream = fs.createReadStream ('file.txt');
    Const writableStream = fs.createWriteStream ('file-copy.txt');

readableStream.pipe (writablyStream);

    readableStream.on ('end', () => {
            Console.log ('file Copied Succesfully'); });
```

4) Event-loop :

- Event loop handles asynchronous Callback, allowing Node.js to perform non-blocking I/O operations.

```
Ex: Console.log ('start');

    SetTimeout (() => {
            Console.log ('Timeout');
            3,000);

    Console.log ('End');
```

5) Child Processes :

- Child Processes enables the execution of Shell Command or other program from a Node.js Script using child_process module.

5. Streams :

Example - Const Spawn = require ('child_process');

Const ls = spawn ('ls', ['-lh', '/usr']);

ls.stdout.on ('data', (data) => {

Console.log (`stdout: ${data}`); });

ls.stderr.on ('data', (data) => {

Console.log (`stderr: ${data}`); });

ls.on ('close', (code) => {

Console.log (`child process exited with code ${code}`); });

6. Cluster Module :-

- Clustering allows you to create child processes that shares server ports, improving performance on multicore system.

7. Debugging And Profiling :

- In Node.js use '--inspect' flag to start debugging and attach a debugger like chrome DevTools or VsCode.

Ex: node --inspect app.js.

- Profiling Tools: Use tools like Node.js built-in profiler, clinic.js and 0x for performance profiling.

3) Streams :

Security :

Best Practices

- Use HTTPS for encrypt Data
- Ensure data validation to prevent Injections.
- Securely save secrets using environment variables in '.env' file.