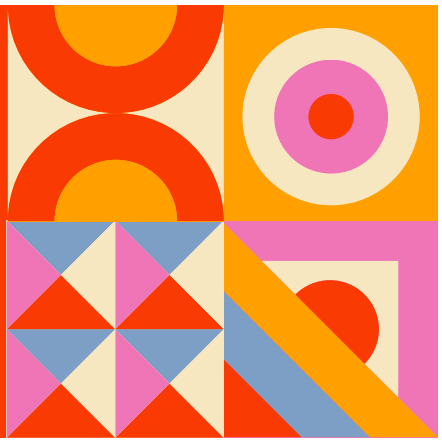
NODE JS



1.What is Node JS ?

- JS Runtime Environment
- Opensource, Cross-platform , and Runtime environment for executing JS code outside a browser
- Before node JS the JS code run on the environment of the browser.
- Node JS is not a programming language and not a framework

2.Features of Node JS

- Cross-platform compatibility
- Asynchronous (Non-blocking)
- Uses JS
- Fast data streaming
- Event driven

ADVANTAGES

FRONT-END BACKEND
HIGH PERFOMANCE
SCALABLE

Cross-Platform : It may be used on variety of systems , including windows , linux ,unix and mobile devices

Asynchronous : A server built with node JS never wait for data from an API

Fast data streaming : When data is transmitted in multiple stream processing them take a long time .Node JS process data at very rate

Event driven : It means as soon as node starts its server simply initiates its variables declare functions and then simply wait for event to occur.

3.Creating and Running basic Node JS Application

- Import required modules
- Create server
- Read request and return responses

Package.json
it is a blueprint or manifest of ou
project

It contails : Project info
Dependecies
Scripts

1.Import required modules

Load Node modules using required directive

Eg : var http = require(“http”)

2.Create server

Create a server to listen to the client request.

createServer() method is used for server creation

Then bind the server to port 8080 using the listen method associated with the server instance

http.createServer().listen(8080);

3.Read request and return responses

A function with request and response parameter is used to read client request and return responses.

http.createServer(function(req,res)=>{

....

}).listen(8080);

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

http.createServer(function (req, res) {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('Congrats you have a created a web server');
    res.end();
}).listen(8080, () => { console.log("Server is running on 8080") });
```

4 . What is module in Node JS ?

A module is a set of JavaScript functions and objects that can be used in an application.

Modules are similar to JavaScript libraries and are the building blocks of Node.js applications.

5 . What are the different types of modules in Node Js ?

1. Core Module / In-built Module
2. Local Module

3. Third party Module / NPM Module

Core Module :

Node.js has many built-in modules that are part of the platform and come with **Node.js** installation.

These modules can be loaded into the program by using the **required** function.

Syntax : const module = require('module_name');

Examples : http, fs, url, path, os

Local Module :

Unlike built-in and external modules, local modules are created locally in your Node.js application.

Create our own modules and we can use it in our project

Third party Module :

Third-party modules are modules that are available online using the Node Package Manager(NPM).

Some of the popular third-party modules are Mongoose, express, angular, and React.

Example :

npm install express

npm install mongoose

6 . Node.js File System (FS)

It is used to handle file operations like creating , reading, deleting etc.

Node js provides an inbuilt module called FS

To use this module we need to require FS at first .

var fs = require('fs')

fs.mkdir()

It is used to create a directory.

Syntax : fs.mkdir(path, mode, callback)

```
// folder create
fs.mkdir(path.join(__dirname,"/Images"),{},(error)=>{
    if (error) {
        console.log(error);
    }
    else{
        console.log("Folder Created Successfully");

    }
})
```

fs.writeFile()

It is used to asynchronously write the specified data to a file . By default the file would be replaced if it exists

```
// write file
fs.writeFile(path.join(__dirname,'/Image','note.txt'),"Name : Megha tp",(error)=>{
    if (error) {
        console.log(error);
    }
    else{
        console.log("Successfully File created");

    }
})
```

fs.appendFile()

It is used to asynchronously append the given data to a file .

```
const user = 'Nihal k'

fs.appendFile(path.join(__dirname,'/Image','abcd.txt'),`\n${user}`,(error)=>{
    if (error) {
        console.log(error);
    }
    else{
        console.log("Successfully File created");

    }
})
```

fs.readFile()

It is used to read the file.

```
fs.readFile(path.join(__dirname,'/Image','abcd.txt'),'utf-8',(error,data)=>{
    if (error) {
        console.log(error);
    }
    else{
        console.log(data);
    }
})
```

fs.unlink()

It is used to delete file

```
// Removing a file named 'note.js'
fs.unlink(path.join(__dirname, "/Image",'note.js'), (error) => {
    if (error) {
        console.log("Error deleting file:", error);
    } else {
        console.log("File deleted successfully");
    }
});
```

fs.rmdir()

It is used to delete a directory at the given path

```
// Removing a directory using fs.rmdir ()

fs.rmdir(path.join(__dirname, '/Imagess'), (error) => {
    if (error) {
        console.log(error);
    } else {
        console.log('Directory removed successfully');
    }
});
```

7 . Node Package Manager

NPM (Node Package Manager) is the default package manager for Node and is written entirely in JavaScript.

Npm Commands :

1. npm install

   Installs a package in the package.json file in the local node_modules folder.

2. npm uninstall

Remove a package from the package.json file and removes the module from the local node_modules folder.

3. npm update

This command updates the specified package.

4. npm init

Creates a package.json file in our directory.

5. npm start

Runs a command that is defined in the start property in the scripts. 6. npm version

8 . Node JS - REPL

REPL (READ, EVAL, PRINT, LOOP) is a computer environment similar to Shell (Unix/Linux) and command prompt.

Node comes with the REPL environment when it is installed.

- Read: It reads the inputs from users and parses it into JavaScript data structure. It is then stored to memory.
- Eval: The parsed JavaScript data structure is evaluated for the results.
- Print: The result is printed after the evaluation.
- Loop: Loops the input command.

9 . Node JS - Timers

setTimeout()

setTimeout() is a built-in function that is used to schedule the execution of a function after a specified delay.

The delay is provided in milliseconds, and once that delay has passed, the specified function is executed.

```
setTimeout(() => {
    console.log("Hello world");
}, 5000);
```

setInterval()

It is a built-in function that is used to repeatedly execute a function at a specified time interval.

```
setInterval(() => {
    console.log("Hello world");
}, 5000);
```

clearInterval()

clearInterval() is used to stop a function that was set to run repeatedly by setInterval().

```
// Define a function that logs a message every second
const intervalId = setInterval(() => {
    console.log('This message repeats every 1 second.');
  }, 1000);

  // Stop the interval after 5 seconds
  setTimeout(() => {
    clearInterval(intervalId);
    console.log('Interval cleared after 5 seconds.');
  }, 5000);
```

setImmediate()

setImmediate() schedules the function to be executed after I/O events are processed, or at the end of the current event loop.

```
console.log('Start');

// Schedule a function to run immediately after the I/O events are processed
setImmediate(() => {
  console.log('hyy.');
});

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

10 . What is Synchronous?

Synchronous programming means that **the code runs in the sequence it is defined**. In a synchronous program, when a function is called and has returned some value, only then will the next line be executed.

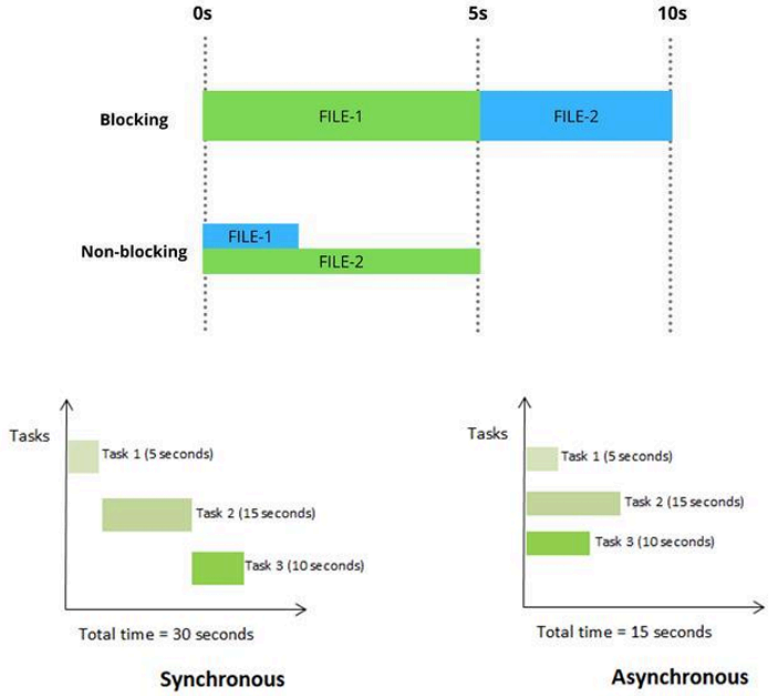
Synchronous code is also called “blocking” as it halts the program until all the resources are available.

11 . What is Asynchronous?

The program doesn’t wait for the task to complete and can move on to the next task.

Asynchronous code is also known as “non-blocking”.

The program continues executing and doesn’t wait for external resources (I/O) to be available.

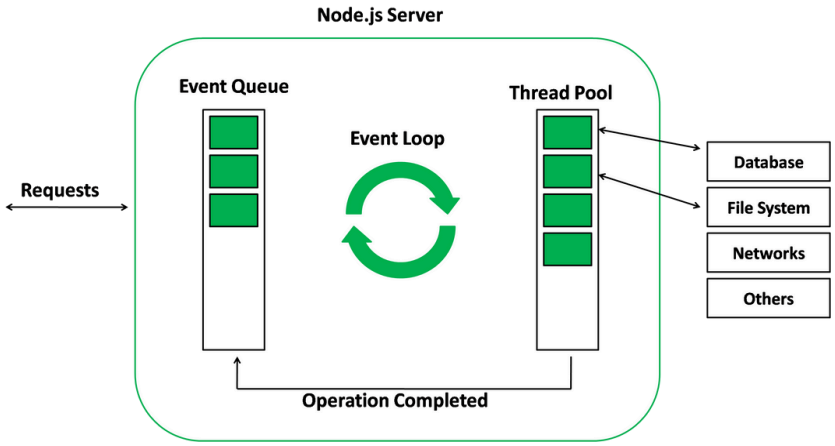


12 . What is the Event Loop?

The **event loop** in Node.js is a fundamental mechanism that allows Node.js to perform non-blocking, asynchronous operations, even though JavaScript is single-threaded.

It continuously checks for tasks, executes them, and ensures that the system doesn't block while waiting for operations like file I/O, network requests, or timers to complete.

13 . Working of the Event loop?



When Node.js starts, it initializes the event loop, processes the provided input script which may make async API calls, schedules timers, then begins processing the event loop.

When using Node.js, a special library module called libuv is used to perform async operations.

This library is also used, together with the back logic of Node, to manage a special thread pool called the libuv thread pool.

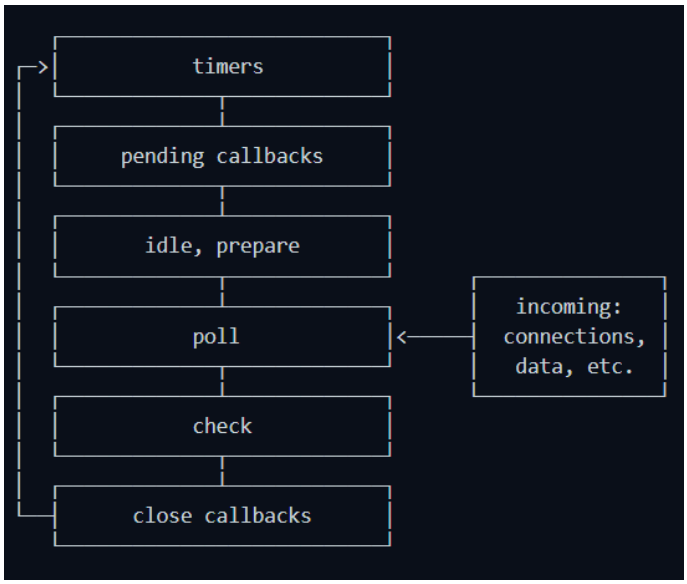
This thread pool is composed of four threads used to delegate operations that are too heavy for the event loop.

I/O operations, Opening and closing connections, setTimeouts are examples of such operations.

When the thread pool completes a task, a callback function is called which handles the error(if any) or does some other operation.

This callback function is sent to the event queue. When the call stack is empty, the event goes through the event queue and sends the callback to the call stack.

14 . Phases of the Event loop:



1. Timers:

This phase processes timers that have been set using setTimeout() and setInterval().

2. Pending Callbacks:

This phase processes any callbacks that have been added to the message queue by asynchronous functions.

3. Idle, Prepare:

The “idle.ignore” phase is not a standard phase of the event loop in Node.js.

It means it’s Used internally only. The “idle” phase is a period of time during which the event loop has nothing to do and can be used to perform background tasks, such as running garbage collection or checking for low-priority events.

4. Poll:

This phase is used to check for new I/O events and process any that have been detected.

5. Check :

This phase processes any setImmediate() callbacks that have been added to the message queue.

6. Close Callbacks:

This phase processes any callbacks that have been added to the message queue by the close event of a socket.

This means that any code that needs to be executed when a socket is closed is placed in the message queue and processed during this phase.

The event loop is a powerful feature of Node.js that enables it to handle a high number of concurrent connections and perform non-blocking I/O operations.

15 . The Node.js Event emitter

If you worked with JavaScript in the browser, you know how much of the interaction of the user is handled through events: mouse clicks, keyboard button presses, reacting to mouse movements, and so on.

On the backend side, Node.js offers us the option to build a similar system using the events module.

This module, in particular, offers the EventEmitter class, which we'll use to handle our events.

You initialize that using

```
const EventEmitter = require('node:events');

const eventEmitter = new EventEmitter();
```

This object exposes, among many others, the on and emit methods.

- emit is used to trigger an event
- on is used to add a callback function that's going to be executed when the event is triggered

```
const EventEmitter = require('node:events');
const eventEmitter = new EventEmitter();

eventEmitter.on('start', () => {
    console.log('started');
  });

eventEmitter.emit('start');
```