**NodeJS Tutorial**

(https://www.geeksforgeeks.org/nodejs/?ref=lbp)

* Node.js is a open-source, and cross-platform JavaScript runtime environment built on [Chrome's V8 engine](https://www.geeksforgeeks.org/explain-v8-engine-in-node-js/).
* It allows you to run JavaScript code outside the browser, making it ideal for building scalable server-side and networking applications.
* NodeJS is single-threaded, utilizing an event loop to handle multiple tasks concurrently.
* It is asynchronous and non-blocking, meaning operations do not wait for execution to complete. Operations execute [concurrently](https://en.wikipedia.org/wiki/Concurrent_computing) and use [callbacks](https://en.wikipedia.org/wiki/Callback_(computer_programming)) to signal completion or failure.
* Node.js is used to create fast running APIs (Application Programming Interfaces) and connecting database to the application.
* It can also be used to create Web-applications (front-end), but it is not recommended.
* Node.js was created by Ryan Dahl in 2009.

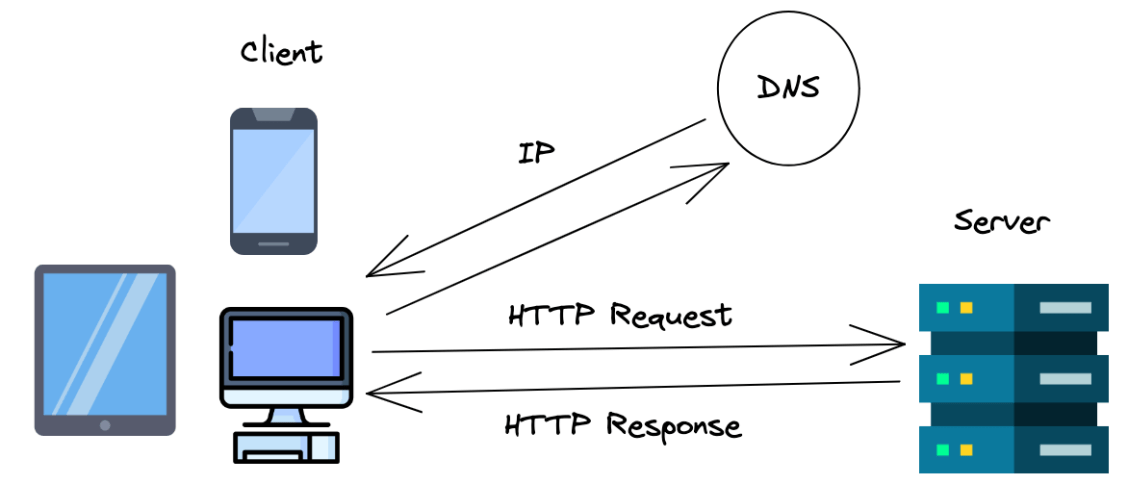
Javascript 🡪 V8 Engine + C++ Program 🡪 Node JS

* JavaScript is the only language that Node.js supports natively, but many compile-to-JS languages such as TypeScript, CoffeeScript are available.

|  |  |  |
| --- | --- | --- |
|  | **Browser Javascript( Vanilla JS)** | **Node.js** |
| **Runs on** | Browser | Server or Local Machine |
| **DOM Manipulation** | Yes | No |
| **Window Object** | Yes | No |
| **Export/import** | Yes | No |
| **Database access** | No | Yes |
| **File System** | No | Yes |

**Note:** The **V8 JavaScript engine** is an **open-source JavaScript engine** developed by Google. It powers the Google Chrome browser and Node.js, enabling high-performance execution of JavaScript by converting it into machine code directly.

**Client-Server Model:** A distributed computing architecture where clients request resources from server and server verifies request and respond with required resources.



**How to use Node.js:**

1. Download and install Node.js
2. Verify Installation: node -v
3. Create a js file, lets say hello.js
4. Run using command: node hello
5. To generate package.json: npm init -y (optional)

The **package.json** file is a central configuration file in Node.js projects. While its primary purpose is to list dependencies (i.e., libraries or modules required for the project), it also provides additional functionality, including:

* + Defining scripts to automate tasks.
  + Specifying metadata about the project, such as the name, version, author, and license.
  + Configuring project-specific settings.

**How to import/export in Node.js**

* Node.js don’t have export and import functionality as we have in React.
* Node.js uses
  + module.export = {field or method name } // for export
  + require(“filename”) // for import

**NPM (Node Package Manager)**:

* It is a standard package manager for NodeJS.
* It is used to install, update, and manage packages and dependencies for Node.js projects.
* By installing NodeJS on your system, NPM is automatically installed, and ready to use.
* Packages are stored in the node\_modules folder in our project.
* **Initialize a NPM project**: npm init
* **Install Packages with NPM:** npm install <package-name>
* **Install a Specific Version:** npm install <package-name>@<version>
* **Install Packages Globally:** npm install -g <package-name>
* **Install All Dependencies listed in the package.json:** npm install
* **Updating Packages:** npm update
* **Uninstalling Packages:** npm uninstall <package-name>
* **package-lock.json** file locks the versions of your project’s dependencies, ensuring consistent installations across different environments.

**Note:** While NPM is the default package manager for NodeJS, there are alternatives like [Yarn](https://www.geeksforgeeks.org/difference-between-npm-and-yarn/) and [pnpm](https://www.geeksforgeeks.org/pnpm-vs-npm/).

**Popular NPM Packages for NodeJS:**

|  |  |
| --- | --- |
| **Packages** | **Description** |
| **Express** | A fast, minimal web framework for building APIs and web applications. |
| **Mongoose** | A MongoDB object modeling tool for NodeJS. |
| **Axios** | A promise-based HTTP client for making HTTP requests. |
| **React** | A popular front-end library used to build user interfaces. |
| **Nodemon** | Automatically restarts the server during development when file changes are detected. |
| **Socket.io** | Enables real-time, bidirectional communication between web clients and servers. |

**Modules in NodeJS**

* NodeJS is built around the concept of [modules](https://www.geeksforgeeks.org/node-js-modules/).
* Modules are reusable pieces of code that can be imported into different parts of an application.
* It helps break down large applications into smaller, manageable sections, each focused on a specific functionality.
* A Module can containvariables, functions, class and objects.

**NodeJS provides two primary module systems:**

1. **ES6 Modules:**

* Uses import to import modules.
* Uses export to export functions, objects, or variables.
* Modules are loaded asynchronously, allowing better performance.
* Requires “type”: “module” in package.json.

Example:

//app.js

import { add, PI } from './math.js';

console.log(add(2, 3\*PI));

//math.js

**export** **function** add(a, b) {

**return** a + b;

}

**export** **const** PI = 3.1415;

1. **CommonJS Modules (CJS):**

* CommonJS is the default module system used in [NodeJS](https://www.geeksforgeeks.org/nodejs/).
* It enables code modularity by allowing developers to export and import functions, objects, or variables using module.exports and require().
* Modules are loaded synchronously, meaning execution waits until the module is fully loaded.

**Example:**

//math.js

function add(a, b) {

return a + b;

}

const PI = 3.1415;

module.exports = {

    add,

    PI

};

//app.js

const {add, PI} = require("./math.js");

console.log(add(2, 3\*PI));

**Types of Modules:**

1. **Local Modules:**

A local module in Node.js refers to a custom module created in an application. Unlike the built in or third-party modules, local modules are specific to the project and are used to organize and reuse your code across different parts of your application.

1. **Built-in Modules:** Modules which are provided with node.js packages.

Examples: HTTP Module, FS Module, Path Module etc.

1. **Third-party Modules:** They are external packages, need to install using NPM.

Example: Express, Mongoose etc.

**Common Built-in NodeJS Modules:**

1. **FS (File System) Module:**

* The fs (File System) module in Node.js provides an API for interacting with the file system.
* It allows you to perform operations such as reading, writing, updating, and deleting files and directories, which are essential for server-side applications and scripts.
* All file system operations can have synchronous and asynchronous forms depending upon user requirements.

**Synchronous Vs Asynchronous approach:**

* 1. **Synchronous approach:**

They are called **blocking functions** as it waits for each operation to complete, only after that, it executes the next operation.

const fs = require("fs");

// Synchronous read

const data = fs.readFileSync('input.txt');

console.log("Synchronous read: " + data.toString());

* 1. **Asynchronous approach:**

They are called **non-blocking functions** as it never waits for each operation to complete, rather it executes all operations in the first go itself. It uses a callback to know the result of a operation.

const fs = require("fs");

// Asynchronous read

fs.readFile("input.txt", (err, data) => {

if (err) return console.error(err);

console.log("Asynchronous read: " + data.toString());

});

**Note:** If your operations are not doing very heavy lifting like querying huge data from DB then go ahead with Synchronous way otherwise Asynchronous way.

1. **Open a File:**

**Syntax:**

fs.open(path, flags, mode, callback);

**Parameters:**

* **path**: It holds the name of the file to read or the entire path if stored at other locations.
* **flags**: Flags indicate the behavior of the file to be opened. All possible values are ( r, r+, rs, rs+, w, wx, w+, wx+, a, ax, a+, ax+). (optional)
* **mode**: Sets the mode of file i.e. r-read, w-write, r+ -readwrite. It sets to default as readwrite. (optional)
* **callback:** It is a callback function that is called after opening of the file. It takes two parameters:
  + **err:** If any error occurs.
  + **fd:** file descriptor.

1. **Reading a File:**

fs.readFile(path, (err, buff\_data)=>{

if(err) throw err.stack

console.log(buff\_data.toString());

});

1. **Writing to a File**: This method will overwrite the file if the file already exists or will create the file if not exists. The fs.writeFile() method is used to asynchronously write the specified data to a file.

fs.writeFile(path, data, options, callback)

* **options:** It is an optional parameter that specifies the encoding (utf-8)/mode/flag.

1. **Appending to a File**: to synchronously append the data to the file.

fs.appendFile(filepath, data, options, callback);

1. **Delete a File:** To remove a file or symbolic link from the filesystem.

fs.unlink(path, callback)

1. **fs.exists():** method in node is used to check if the input file or the directory exists or not.

It returns **boolean** values which signifies that the path exists or not.

fs.exists('/etc/passwd.js', (exists) => {

console.log(exists ? 'Found' : 'Not Found!');

});

**Note:** It is **now deprecated**, and it is recommended to use **fs.access()**

1. **fs.mkdir() method** in Node.js is used to create a directory asynchronously.

fs.mkdir(path, mode, callback)

1. **fs.renameSync()** method is part of the built-in File System (fs) module and is used to rename or move files and directories synchronously.

fs.renameSync( oldPath, newPath )

1. **fs.rmdir() method** is used to delete a directory at the given path. It can also be used recursively to remove nested directories.

fs.rmdir( path, options, callback )

1. **OS module:**

It provides operating system-related utility methods and properties. It helps retrieve system information such as CPU details, memory usage, and network interfaces, enabling you to write system-aware applications.

const os = require(“os”);

**Some useful methods:**

* **os.arch():**Returns the CPU architecture of the operating system (e.g., ‘x64’, ‘arm’).
* **os.cpus():**Provides an array of objects describing each CPU/core installed.
* **os.freemem():**Returns the amount of free system memory (RAM) in bytes.
* **os.totalmem():**Returns the total amount of system memory (RAM) in bytes.
* **os.homedir():**Returns the path to the current user’s home directory.
* **os.hostname():**Returns the hostname of the operating system.
* **os.networkInterfaces():**Returns a list of network interfaces and their details.
* **os.platform():**Returns the operating system platform (e.g., ‘linux’, ‘darwin’, ‘win32’).
* **os.release():**Returns the operating system release.
* **os.uptime():**Returns the system uptime in seconds.
* **os.userInfo():** Returns the information of the current user.

1. **Path Module:**

It provides the utilities for working with file and directory paths. It allows for path manipulation, normalization, resolution, and parsing of file paths, making it easier to handle file system paths across different operating systems.

const path = require(“path”);

**Some useful methods:**

* **path.basename(“Relative Path of the file”):** Returns the filename portion of a path to the file.
* **path.dirname(“Relative Path”):** Returns the directory name of the given path.
* **path.extname(“Relative Path”):** Returns the extension portion of a file path.
* **path.join(path1, path2, ..):** joins a number of path segments using the platform-specific delimiter (e.g., a slash / on Linux and macOS and a backslash \ on Windows) to form a single path.
* **path.parse(path):** Returns an object whose properties represent the given path. Object contains the following properties:
  + root (root name)
  + dir (directory name)
  + base (filename with extension)
  + ext (only extension)
  + name (only filename)
* **path.resolve(path1, path2, ..):** Resolves a sequence of path segments to an absolute path.

1. **URL Module:**

* The ‘url’ module is used to parse the URL and its properties.
* Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties.

url.parse( urlString, parseQueryString, slashesDenoteHost)

* **urlString:** It holds the URL string which needs to parse.
* **parseQueryString:** It is a boolean value. If it set to true then the query property will be set to an object returned by the querystring module’s parse() method. If it set to false then the query property on the returned URL object will be an unparsed, undecoded string. Its default value is false.
* **slashesDenoteHost:** It is a boolean value. If it set to true then the first token after the literal string // and preceding the next / will be interpreted as the host. For example: //www.google.com/web contains the result {host: ‘www.google.com’, pathname: ‘/web’ } rather than {pathname: ‘//www.google.com /web’}. Its default value is false.

**Note:** 

* If urlString is not a string then it threw TypeError.
* If auth property exists but not decoded then it threw URIError.

**Example:**

var url = require('url');  
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';  
var q = url.parse(adr, true);  
console.log(q.host); //returns 'localhost:8080'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?year=2017&month=february'  
var qdata = q.query; //returns an object: { year: 2017, month: 'february' }  
console.log(qdata.month); //returns 'february'

1. **EVENTS Module:**

Event is basically a signal that something has happened in the application.

Event emitter is used to create custom events and react to those events whenever they are emitted or called.

In this module, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, need to create an EventEmitter object.

**Methods:**

* **on()** method is used to register an event listener in Node.js. It attaches a handler function to an event that is emitted on an EventEmitter object.
* **emit()** is used to emit an event multiple times, triggering all the listeners registered for that event.
* **once()** registers a listener that will only be executed once. After the first invocation, the listener is automatically removed, so it won’t be triggered again.
* **removeListener()** is the method used to remove an event listener from an EventEmitter object.

**Example:**

const events = require('events');

const eventEmitter = new events.EventEmitter();

//Create an event handler

function onClickHandler(){

    console.log("Button clicked");

}

//Register the event handler to an event

eventEmitter.on("click", onClickHandler);

//Trigger the 'click' event

eventEmitter.emit('click');

1. **Stream Module:**

* Streams in Node.js are primarily used for handling asynchronous input/output (I/O) operations.
* They allow you to read or write data in chunks sequentially, enabling more efficient handling of large amounts of data without needing to load it all into memory at once.
* According to Node.js documentation, “*A stream is an abstract interface for working with streaming data in Node.js”.*

const stream = require('stream');

**Types of Streams in Node.js:**

1. **Writable:** We can write data to these streams. **e.g.,**fs.createWriteStream().
2. **Readable:** We can read data from these streams. **e.g.,** fs.createReadStream().
3. **Duplex:** Streams that are both, Writable as well as Readable. **e.g.,**net.socket.
4. **Transform:** Streams that can modify or transform the data as it is written and read. **e.g.,**zlib.createDeflate.

**Methods:**

* **pipe()** method is used to pipe data from one stream to another.

1. **HTTP module**

* HTTP module is a core built-in module that is used to create and manage HTTP servers, allowing us to handle incoming HTTP requests and send HTTP responses.
* Supports various HTTP methods like GET, POST, PUT, DELETE, etc. by examining the req.method property.

**Creating Servers using HTTP:**

const http = require("http");

//create a server object

const server = http.createServer((req,res)=>{

    res.write("Hello, Server is on"); //write a response to the client

    res.end(); //end the response

});

server.listen(7000); //the server object listens on port <http://localhost:7000>

**Methods:**

* **http.createServer()** method in Node.js is used to create an HTTP server. It takes a callback function that handles incoming requests and sends responses.
* **listen()** is used to actually start the server, binding it to a specified port and making it listen for incoming connections.
* **res.writeHead()** is used to send an HTTP response header to the client, specifying the status code and other header information.
* **res.end()** is used to signal the end of the response and send the actual content to the client.