NodeJs Tutrorial The NetNinja

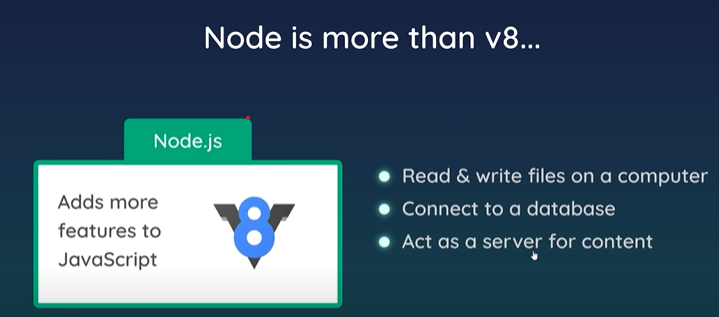
1. Introduction
   1. Introduction

Node.js est une plateforme logicielle open-source basée sur le moteur JavaScript V8 de Google(**V8 aussi nous aide a compiler le code Js sur le navigateur**). Elle permet l'exécution de code JavaScript côté serveur, permettant ainsi de développer des applications web et des services réseau performants et évolutifs.

Une image contenant diagramme

Description générée automatiquement

V8 nous aide a run le code Js sur notre serveur ou notre computer



Node.js a un avantage unique car des millions de développeurs frontaux qui écrivent du JavaScript pour le navigateur sont maintenant capables d'écrire le code côté serveur en plus du code côté client sans avoir à apprendre un langage complètement différent.

Dans Node.js, les nouvelles normes ECMAScript peuvent être utilisées sans problème, car vous n'avez pas besoin d'attendre que tous vos utilisateurs mettent à jour leurs navigateurs - vous avez la charge de décider quelle version ECMAScript utiliser en changeant la version de Node.js, et vous pouvez également activer des fonctionnalités expérimentales spécifiques en exécutant Node.js avec des drapeaux

* 1. Setup
     1. Pourquoi NodeJs

Une image contenant texte

Description générée automatiquement

* + 1. In this Course

Une image contenant texte

Description générée automatiquement

* + 1. Before you start

Une image contenant texte

Description générée automatiquement

1. NodeJs Basic
2. Global Object

Inside the browser the window object is the global object Example:(In the browser we can use set imeout inside window.setTimeout because Window is the global object )

In Node the global object is : global

* 1. Some function and properties and variable and …. In the Global object
* setTimeout
* setInterval
* \_\_dirname
* \_\_filename

// Global Object

global.setTimeout(()=>{

    console.log("in the timeout1")

},3000)

setTimeout(()=>{

    console.log("in the timeout2")

    // faite stopper l'interval int

    clearInterval(int)

},4000)

const int= setInterval(() => {

    console.log("in the interval")

}, 1000);

// Full absolute pth for this directory

console.log(\_\_dirname)

//Full absolute path for this fileName

console.log(\_\_filename)

Une image contenant texte

Description générée automatiquement

1. require

In Node.js, the require() function is used to import modules, which are separate pieces of code that can be reused in multiple files or projects.

When a module is required using require(), Node.js searches for the corresponding file on the file system and loads it into the current script. This allows the module to be used and its functions, objects or variables to be accessed from within the script.

People.js

const people = ["Houssam", "Mohcine", "Hamza", "Salah", "Yasser"];

const ages = [20, 25, 30, 35];

module.exports = { people, ages };

Modules.js

const xyz=require('./people')

const {people} =require('./people')

console.log("Test1",xyz)

console.log("Test2",people)

Une image contenant texte

Description générée automatiquement

1. the File System
   1. fs

In Node.js, the built-in **fs** (file system) module provides a way to work with the file system on the computer running the Node.js application.

Some of the most common uses of the **fs** module in Node.js include:

1. Reading and writing files: The **fs** module provides functions for reading and writing files, including creating, deleting, moving, and copying files.
2. Working with directories: The **fs** module includes functions for working with directories, such as creating and deleting directories, and listing the contents of a directory.
3. Watching for file changes: The **fs** module includes functions for watching files and directories for changes, allowing you to react to changes in real-time.
4. Changing file permissions: The **fs** module provides functions for changing file permissions, such as read, write, and execute permissions.

The most useful Function in fs

In Node.js, the **fs** (file system) module provides many useful functions for working with files and directories. Here are some of the most commonly used functions:

1. **fs.readFile()** - Asynchronously reads the entire contents of a file.
2. **fs.writeFile()** - Asynchronously writes data to a file, replacing the file if it already exists.
3. **fs.appendFile()** - Asynchronously appends data to a file, creating the file if it does not exist.
4. **fs.existsSync()** - Synchronously tests whether a file or directory exists.
5. **fs.mkdir()** - Asynchronously creates a new directory with the specified name.
6. **fs.readdir()** - Asynchronously reads the contents of a directory.
7. **fs.stat()** - Asynchronously retrieves information about a file or directory.
8. **fs.watch()** - Watches a file or directory for changes.
9. **fs.createReadStream()** - Creates a readable stream for a file.
10. **fs.createWriteStream()** - Creates a writable stream for a file.
11. **fs.rmdir()** – Async Remove a directory
12. **fs.unlink()** – Async Remove a file

These are just a few of the many functions provided by the **fs** module. The most useful function will depend on the specific requirements of your application. However, **fs.readFile()** and **fs.writeFile()** are two of the most commonly used functions in Node.js, as they provide a simple way to read from and write to files asynchronously.

const fs =require('fs')

//  Reading Files

// In Node.js, fs.readFile() is an asynchronous function for reading files, which means that it does not block the execution of the rest of the code while waiting for the file to be read.

fs.readFile('./docs/blog1.txt',(err,data)=>{

    if(err){

        console.log("err",err)

    }

    console.log(data.toString())

})

console.log('Last Line')

// Writing Files

// il va modifier le fichier blog1.txt et mettre hello world Async

fs.writeFile('./docs/blog1.txt','hello world',()=>{

    console.log('file was written')

})

// il va cree le fichier blog2.txt car il existe pas puis il va mettre hello again Async

fs.writeFile('./docs/blog2.txt','hello again',()=>{

    console.log('file was written')

})

// Directories

//  existsSync ==>verifie si le fichier ou le dossier exist Syncrone

if(!fs.existsSync('./assets')){

    // cree un dossier Async

    fs.mkdir('./assets',(err)=>{

        if(err){

            console.log("err",err)

        }

        console.log('Folder created')

    })

} else{

    // Supprimer un dossier Async

    fs.rmdir('./assets',(err)=>{

        if(err){

            console.log("err",err)

        }

        console.log('Folder deleted')

    })

}

// deleting files

if(fs.existsSync('./docs/deleteme.txt')){

    fs.unlink('./docs/deleteme.txt',(err)=>{

        if(err){

            console.log(err)

        }

        console.log('file deleted')

    })

}

* 1. Streams & Buffers

In Node.js, streams are a way of handling continuous data flow in a more efficient and scalable way. Streams allow data to be read or written in chunks, rather than loading it all into memory at once. This is particularly useful for working with large amounts of data or for handling data that is being received or sent over a network.

There are four types of streams in Node.js:

1. Readable: A readable stream represents a source of data from which data can be read. Examples include reading data from a file or receiving data from a network socket.
2. Writable: A writable stream represents a destination to which data can be written. Examples include writing data to a file or sending data over a network socket.
3. Duplex: A duplex stream is both readable and writable. Examples include sending and receiving data over a network socket at the same time.
4. Transform: A transform stream is a type of duplex stream that can modify or transform data as it is read from or written to a source or destination.

const fs = require("fs");

const readStream = fs.createReadStream("./docs/blog3.txt", {

  encoding: "utf8",

});

// ReadStream

readStream.on("data", (chunk) => {

  console.log("================= New Chunk ==================================\n\n\n\n\n\n");

  console.log(chunk);

});

// Write stream

const writeStream=fs.createWriteStream('./docs/blog4.txt')

// readStream.on("data", (chunk) => {

//     console.log("================= New Chunk ==================================\n\n\n\n\n\n");

//     console.log(chunk);

//     writeStream.write('\n New Chunk \n')

//     writeStream.write(chunk)

//   });

//   Pipe do the same thing

readStream.pipe(writeStream)