Node.js : Fonaments

Node.js és un entorn d'execució de JavaScript basat en el motor V8 de Google Chrome que permet executar codi JavaScript fora del navegador, principalment per al backend. Això vol dir que pots utilitzar JavaScript, que habitualment s'utilitza en el frontend, també per crear aplicacions del costat del servidor, gestionar bases de dades i desenvolupar APIs. És asíncron, basat en esdeveniments, i altament escalable, cosa que el fa ideal per a aplicacions web modernes.

Per al desenvolupament del codi font, utilitzarem l'editor Visual Studio Code. És un editor creat per Microsoft, gratuït i de codi obert.

S'utilitza sovint amb Node.js per desenvolupar aplicacions backend. Visual Studio Code ofereix integració directa amb Node.js, permetent executar scripts, depurar codi, gestionar paquets amb npm i accedir a extensions específiques per a JavaScript i Node.js.

Instal·lar Node.js

Accedim a la pàgina oficial de Node.js: https://nodejs.org/en

Ens descarreguem l'última versió LTS de Node.js. LTS significa Long Term Support (suport a llarg termini). Una versió LTS és una versió del software que rep actualitzacions de seguretat i manteniment durant un període de temps més llarg, normalment dos o més anys. Són estables i recomanades per a aplicacions en producció, ja que tenen un cicle de vida més previsible i són més segures que les versions regulars.

Per verificar que Node.js s'ha instal·lat correctament, executa aquesta comanda al terminal per veure'n la versió:

```
node -v
```

També s'instal·larà npm (Node Package Manager), que t'ajudarà a gestionar paquets i dependències. Executa la següent comanda al terminal per veure'n la versió:

```
npm -v
```

Executar JavaScript a Node.js

Crea un fitxer JavaScript per executar-lo a Node.js:

Crea un nou directori. Després, dins aquest directori, crea un fitxer anomenat app.js amb el següent contingut:

```
console.log('Hola, món!');
```

Per executar el codi, obre la terminal al directori del projecte, o navega fins al directori, i executa:

```
node app.js
```

Això mostrarà "Hola, món!" al terminal.

O ho podem fer des de Visual Studio Code prement «Run» i seleccionant «Node.js».

Exportar i importar mòduls

«require» forma part del sistema de mòduls CommonJS, que s'utilitza per importar mòduls (fitxers o llibreries) dins d'un projecte Node.js.
Aquest sistema és integrat a Node.js per gestionar dependències i estructurar el codi en diferents fitxers.
És útil separar el codi en diferents fitxers per millorar la organització i facilitar-ne el manteniment.

Definim la funció «saludar» al fitxer «funcions.js» i fem una exportació simple perquè pugui ser importada com a mòdul i usada en altres fitxers:

```
function saludar(nom) {
  return 'Hola, ' + nom + '!';
}
module.exports = saludar;
```

Importem la funció definida a l'altre fitxer:

```
const saludar = require('./funcions');
console.log(saludar('Joan'));
```

Quan utilitzem require, executem el codi del fitxer importat. Si el fitxer no exporta funcions o dades que necessitem utilitzar directament, no cal guardar el retorn; simplement importem el fitxer per executar el seu codi.

Exemple complet amb exportació múltiple (diverses funcions):
https://github.com/xbaubes/DesenvolupamentWeb/tree/main/Backend/Node.js/require

Alguns mòduls built-in interessants

Aquests mòduls venen preinstal·lats a Node.js, però cal importar-los per poder utilitzar-los.

- **Mòdul «fs» : File System**

  Permet treballar amb el sistema de fitxers. Permet llegir, escriure, modificar, eliminar o veure informació de fitxers i directoris.

  Sovint usarem «fs/promises», el mòdul que proporciona una API basada en promeses en lloc de «fs», basada en callbacks, per ser més intuïtiva d'usar. D'aquesta manera totes les funcions retornen una Promise. Si el mòdul ens ofereix una versió únicament en callbacks, podem usar «util.promisify» per obtenir una versió que utilitzi promeses.

  Documentació amb totes les funcions del mòdul:
  https://www.geeksforgeeks.org/node-js-file-system-complete-reference/

  Exemple d'ús de les funcions amb callbacks:
  https://github.com/xbaubes/DesenvolupamentWeb/blob/main/Backend/Node.js/fs/fsFuncions.js

  Exemple d'ús de les funcions amb promeses:
  https://github.com/xbaubes/DesenvolupamentWeb/blob/main/Backend/Node.js/fs/fsPromesesFuncions.js

- **Mòdul «events»**

  Els esdeveniments en Node.js serveixen per gestionar de manera eficient i reactiva accions o canvis en el sistema. Node.js segueix un model d'I/O no bloquejant, on les operacions es fan de forma asíncrona. Els esdeveniments són una manera clau de notificar al programa quan una operació ha finalitzat o quan alguna cosa ha passat.

  En el següent exemple tenim un sistema de gestió de comandes per una botiga en línia. Quan es processa una comanda, volem enviar una notificació, ho farem usant events.

  Importar el mòdul events i crear un objecte «EventEmitter»:

```
const EventEmitter = require('events');
const orderEvents = new EventEmitter();
```

  Escoltar esdeveniments (event listener) amb el mètode «on» de l'objecte EventEmitter:
  on(event, listener): Afegeix un escoltador per a un esdeveniment.

```
function onOrderProcessed(order) {
  console.log(`Order processed: ${order.id}. Total: ${order.total} EUR.`);
}
orderEvents.on('orderProcessed', onOrderProcessed);
```

Emetre esdeveniments amb el mètode «emit» de l'objecte EventEmitter:
emit(event, [...args]): Emet un esdeveniment i passa arguments als escoltadors.

```
const order = { id: 123, total: 299.99 };
orderEvents.emit('orderProcessed', order);
```

Eliminar escoltador amb el mètode «removeListener» de l'objecte EventEmitter:
removeListener(event, listener): Elimina l'escoltador indicat.

```
orderEvents.removeListener('orderProcessed', onOrderProcessed);
```

Documentació amb tots els mètodes de l'objecte EventMitter:
https://nodejs.org/api/events.html#class-eventemitter

Exemple d'ús d'EventMitter:
https://github.com/xbaubes/DesenvolupamentWeb/tree/main/Backend/Node.js/events

- **Mòdul «path»**

Proporciona utilitats per treballar amb rutes de fitxers i directoris. És especialment útil quan es necessita construir, normalitzar, o manipular rutes en un projecte, ja que gestiona les diferències entre els diferents sistemes operatius (per exemple, les rutes en Windows utilitzen \ mentre que en Unix i macOS utilitzen /).

path.join([...paths]): Uneix segments de rutes en una sola. Automàticament afegeix els separadors necessaris i resol .. i .

```
const fullPath = path.join('/users', 'john', '..', 'admin', 'file.txt');
console.log(fullPath); // En Windows: \users\admin\file.txt
```

path.resolve([...paths]): Resol una seqüència de segments de ruta en una ruta absoluta.
Si algun segment comença amb /, es considera com a ruta absoluta i la resta es descarta.

```
const absolutePath = path.resolve('folder', 'subfolder', 'file.txt');
console.log(absolutePath); // En Linux: /home/john/folder/subfolder/file.txt'
```

path.basename(path, [ext]): Retorna el nom del fitxer de la ruta donada (incloent l'extensió).
Si es proporciona un segon paràmetre, aquest extreu l'extensió.

```
const filename = path.basename('/users/john/file.txt');
console.log(filename); // 'file.txt'
```

path.dirname(path): Retorna el directori pare d'una ruta.

```
const dir = path.dirname('/users/john/file.txt');
console.log(dir); // En Linux: '/users/john'
```

path.extname(path): Retorna l'extensió d'un fitxer.

```
const ext = path.extname('/users/john/file.txt');
console.log(ext); // '.txt'
```

Documentació completa de tots els mètodes:
https://nodejs.org/api/path.html

- **Mòdul «crypto»**

  Proporciona funcionalitats de criptografia, incloent hashing, xifratge, generació de claus i signatures digitals. Permet gestionar la integritat i la confidencialitat de les dades, així com crear funcions aleatòries segures. És una eina essencial per implementar mesures de seguretat en aplicacions de Node.js.

  El xifrat és un procés que transforma dades llegibles (text pla) en una forma inintel·ligible (text xifrat) mitjançant un algoritme i una clau. Això es fa per protegir la informació de l'accés no autoritzat.

  Conceptes:

  - ➔ Clau: Una cadena de bytes utilitzada per a l'algoritme de xifrat.
  - ➔ IV (Vector d'Inicialització): Un valor aleatori que s'utilitza juntament amb la clau per a garantir que el mateix text pla generi diferents textos xifrats.
  - ➔ Algorisme de Xifrat: L'algoritme utilitzat per a xifrar i desxifrar dades.

  `crypto.createHash(algoritme)`: Crea un objecte hash que utilitza un algorisme especificat (sha256, md5, etc.) per generar un hash de dades. El hashing és una tècnica utilitzada per transformar dades en una cadena de longitud fixa que representa aquestes dades.

```
const hash = crypto.createHash('sha256');
hash.update('text per hash'); // Afegim el text al que aplicarem el hash
const result = hash.digest('hex'); // Generem el hash en format hexadecimal
```

  Posteriorment l'apliquem a un text.

  `crypto.randomBytes(size)`: Genera un buffer de bytes aleatoris de la mida especificada, que és útil per generar claus i IV.

```
const key = crypto.randomBytes(32); // Genera una clau de 256 bits (32 bytes)
```

  `crypto.createCipheriv(algoritme, clau, iv)`: Crea un objecte cipher per xifrar dades amb un algorisme, una clau i un vector d'inicialització (IV).

```
const cipher = crypto.createCipheriv('aes-256-cbc', key, iv);
// Xifra el text des del format UTF-8 i el converteix a hexadecimal
let encrypted = cipher.update('text a xifrar', 'utf8', 'hex');
encrypted += cipher.final('hex');
```

  Posteriorment l'apliquem a un text.

  `crypto.createDecipheriv(algoritme, clau, iv)`: Crea un objecte decipher per desxifrar dades que han estat xifrades amb un algorisme, una clau i un IV.

```
const decipher = crypto.createDecipheriv('aes-256-cbc', key, iv);
// Desxifra les dades xifrades en format hexadecimal 'encryptedData'
// Converteix les dades xifrades a UTF-8
let decrypted = decipher.update(encryptedData, 'hex', 'utf8');
decrypted += decipher.final('utf8');
```

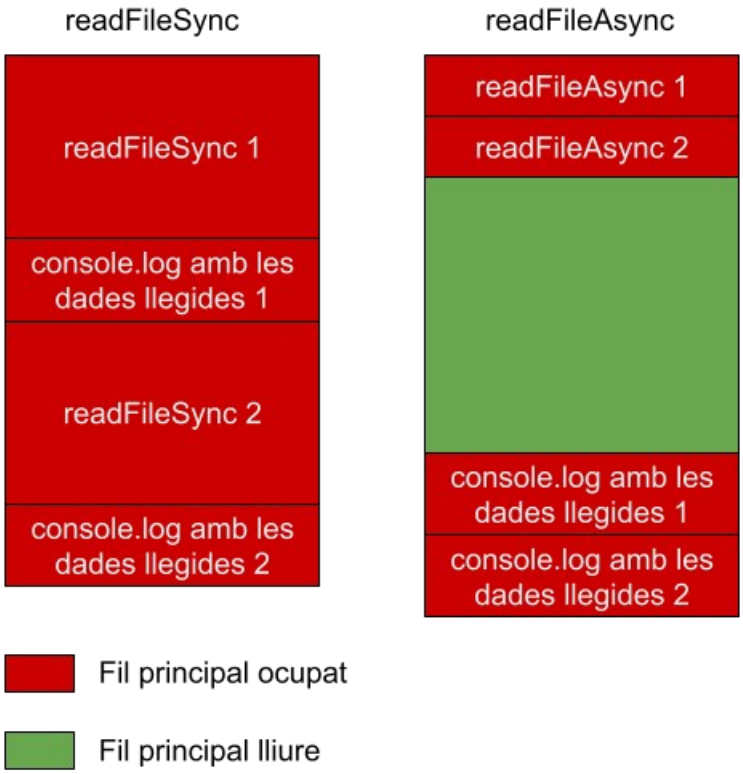
  Posteriorment l'apliquem a un text xifrat.

Revisa el següent exemple:
https://github.com/xbaubes/DesenvolupamentWeb/blob/main/Backend/Node.js/crypto/basicExample.js
En aquest exemple, es fa servir algorisme de xifratge simètric que utilitza l'AES (Advanced Encryption Standard) amb un tamany de clau de 256 bits i un mode de funcionament CBC (Cipher Block Chaining): AES-256-CBC.

Asincronisme

L'asincronisme permet executar múltiples tasques al mateix temps sense bloquejar l'execució del codi. Això vol dir que les operacions que poden trigar, com les consultes a bases de dades o les sol·licituds de xarxa, poden realitzar-se en segon pla mentre el programa continua executant altres tasques. Això millora l'eficiència i la resposta de les aplicacions.

Diferència entre sincronisme i asincronisme:



Codi que mostra el funcionament de l'asincronisme usant el mòdul «fs»:
https://github.com/xbaubes/DesenvolupamentWeb/tree/main/Backend/Node.js/SyncVsAsync

- **Promise**
  És un objecte que representa una operació asíncrona que pot completar-se amb èxit o amb error en el futur. «resolve» i «reject» són funcions proporcionades automàticament per JavaScript quan es crea la promesa, es cridarà una o l'altra segon si la operació ha finalitzat amb èxit o no.

Funció asíncrona

```
function AsyncFunction() {
    return new Promise((resolve, reject) => {
        if ("Operació es resol correctament") {
          resolve('Operació exitosa');
        } else {
          reject('Operació fallida');
        }
    });
}
```

Crida funció asíncrona

```
AsyncFunction()
    .then(result => {
      console.log(result);
    })
    .catch(error => {
      console.log(error);
    })
    .finally(() => {
      console.log('Operació acabada');
    });
```

- **async**
  La funcio async s'inicia i retorna immediatament una Promise mentre la funció s'executa en segon pla el codi continua, ja que no el bloqueja com ho fa el codi síncron.
  Aquesta funció permet l'ús de await per gestionar operacions asíncrones dins seu.

- **await**
  Només es pot usar dins una funció async, pausa l'execució del codi fins que la Promise que està esperant es resol o es rebutja. Això facilita la lectura i la gestió del codi asíncron, ja que li dóna un aspecte més síncron.

Codi que mostra el funcionament de les promeses:
https://github.com/xbaubes/DesenvolupamentWeb/blob/main/Frontend/JavaScript/exampleCode/Promise.js

Codi que mostra el tractament alternatiu de les promeses amb async/await:
https://github.com/xbaubes/DesenvolupamentWeb/blob/main/Frontend/JavaScript/exampleCode/AsyncAwait.js

ACTIVITATS

En aquestes activitats hauràs de combinar alguns dels mòduls que has après.
Gestiona els possibles errors.

1. **Cronòmetre amb events**

   El sistema ha de tenir un comptador que s'incrementa cada segon. Pots usar: setInterval().
   Cada vegada que el comptador s'incrementa, emet un esdeveniment que mostra el nou valor.
   Quan el comptador arribi a un determinat valor, ha d'emetre un esdeveniment que indiqui que el comptador ha arribat al seu límit i parar-se.

   El cronòmetre s'inicia i obté els valors de configuració des dos formes alternatives:

   A través d'un JS Object:

   ```
   let instruccions = {
     maxim : 10,
     msgSegon : "Comptador actual: ",
     msgFinal : "El cronòmetre ha arribat a "
   };
   ```

   A través de paràmetres de consola:

   ```
   node listener.js
   10
   "Comptador actual: "
   "El cronòmetre ha arribat a "
   ```

   Utilitza process.argv per recollir els paràmetres d'entrada per consola en forma d'array.
   Si l'entrada no conté 5 valors (node + listener.js + 3 valors) retorna error. Els paràmetres es guarden com strings, si és necessari, pots fer la conversió a enter amb «parseInt()».

   Això és el que s'ha de mostrar a través de la terminal, al cap de 5 segons i al final:

   ```
   Comptador actual: 1
   Comptador actual: 2
   Comptador actual: 3
   Comptador actual: 4
   Comptador actual: 5
   ```

   Al cap de 5 segons

   ```
   Comptador actual: 1
   Comptador actual: 2
   Comptador actual: 3
   Comptador actual: 4
   Comptador actual: 5
   Comptador actual: 6
   Comptador actual: 7
   Comptador actual: 8
   Comptador actual: 9
   Comptador actual: 10
   El cronòmetre ha arribat a 10
   ```

   Distribuirem el codi en els següents fitxers:

   - **counter.js**: Des d'aquí es compten els segons i s'envien els events.
   - **listener.js**: Des d'aquí es gestionen els events i els valors de configuració, sigui quin sigui el mètode d'entrada. Defineix la funció «configuration()» que inicia l'app.
   - **index.js**: Des d'aquí provem l'app amb la inicialització a través de JS Object.

2. Xifratge i desxifratge de fitxers

Pas 0: Crear els fitxers JS

- **key.js:** Conté la clau i l'algorisme que usaràs per el xifratge i el desxifratge.
- **encrypt.js:** Implementa la funcionalitat de xifratge.
- **decrypt.js:** Implementa la funcionalitat de desxifratge.
- **encryptExample.js:** Serveix com a exemple de com utilitzar encrypt.js amb un fitxer.
- **decryptExample.js:** Serveix com a exemple de com utilitzar decrypt.js amb un fitxer.

Pas 1: Implementació de clau i algorisme

- **key.js:**
  - Defineix una clau secreta de 32 bytes que utilitzaràs per al xifratge. Un cop creada no ha de canviar, sinó no es podrien desxifrar els fitxers. Per major seguretat, en un projecte a producció, la clau hauria d'estar emmagatzemada en variables d'entorn, enlloc d'estar al codi font.
  - Defineix l'algorisme de xifratge que utilitzaràs.

Pas 2: Implementació del xifratge

- **encrypt.js:**
  Importa el mòdul crypto i el fitxer key.js.

  ```
  function encryptData(data)
  ```
  Encripta les dades, usant les claus i l'algorisme especificats i generant un vector d'inicialització (IV) de 16 bytes. Es genera un IV aleatori per a cada xifrat.
  - Rep un JS Object.
  - Genera i retorna un objecte amb el següent format:
    ```
    { iv: , encryptedData: }
    ```
    Aquest objecte conté l'IV generat per xifrar les dades i les dades xifrades. Hauràs d'usar aquest IV per desxifrar les dades xifrades. L'IV és un objecte Buffer, per fer-lo més llegible el guardaràs en format hexadecimal, investiga com fer la conversió amb «toString()» per realitzar aquesta tasca.

Pas 3: Implementació del desxifratge

- **decrypt.js:**
  Importa el mòdul crypto i el fitxer key.js.

  ```
  function decryptData(encryptedData, iv)
  ```
  Desencripta les dades usant la clau i l'algorisme especificats i l'IV rebut.
  - Rep les dades xifrades i un IV en format bytes per desxifrar-lo.
  - Retorna les dades desxifrades en format JS Object.

Pas 4: Exemple d'ús

- **encryptExample.js:**

  Utilitza la funcionalitat d'encrypt.js per xifrar un JS Object.
  Defineix un objecte amb dades de prova (per exemple, un nom, edat i correu electrònic) i executa la funció xifratge.

  Un cop usada la funció encryptData, converteix el text xifrat retornat a JSON i guarda'l en un fitxer anomenat data.json.
  El contingut del fitxer tindrà el següent format:
  ```
  { "iv": , "encryptedData": }
  ```

  Mostra un missatge indicant que les dades s'han xifrat correctament.

- **decryptExample.js:**
  Utilitza la funcionalitat de decrypt.js per desxifrar les dades del fitxer data.json creat prèviament.

  ```
  function readAndDecryptFile(filePath)
  ```
  Gestiona el desxifratge del fitxer rebut.
  - Rep un fitxer en format JSON. Assegura't abans d'intentar obrir-lo que el fitxer rebut realment existeix.
  - Converteix les dades llegides del fitxer de JSON a JS Object. El fitxer conté el text xifrat i un IV per desxifrar-lo en format hexadecimal.
  - Utilitza la funció decryptData per desxifrar el text. Li passem el text xifrat i l'IV en format bytes, investiga com fer la conversió amb: «Buffer.from()».

  Mostra un missatge mostrant les dades desxifrades.

Pas 5: Execució

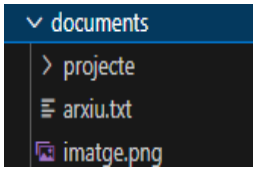
Executa **encryptExample.js** per crear el fitxer data.json amb les dades xifrades.
Després, executa **decryptExample.js** per llegir i desxifrar les dades.

3. Anàlisi del contingut d'una carpeta

Crea una funció que llegeixi tots els fitxers dins d'una carpeta. A la funció li hem de passar la ruta relativa de la carpeta a analitzar a partir de la ruta del fitxer que s'està executant.
La funció ha de retornar la següent informació de cada element (fitxer o subdirectori) dins d'aquesta carpeta:

Fitxer	Subdirectori
Nom base	Nom base
Ruta absoluta	Ruta absoluta
Nom del directori on es troba	Nom del directori on es troba
Extensió	
Mida	

Per exemple, si indiquem que ho faci per aquesta carpeta «documents» retornarà, en aquest format, la següent informació:




```
(3) [{…}, {…}, {…}]
```

(index)	name	absolutePath	dirName	type	extension	size
0	'arxiu.txt'	'D:\\temari-repositori-github\\nodeJS\\documents\\arxiu.txt'	'D:\\temari-repositori-github\\nodeJS\\documents'	'file'	'.txt'	148
1	'projecte'	'D:\\temari-repositori-github\\nodeJS\\documents\\projecte'	'D:\\temari-repositori-github\\nodeJS\\documents'	'subdirectory'		
2	'imatge.png'	'D:\\temari-repositori-github\\nodeJS\\documents\\imatge.png'	'D:\\temari-repositori-github\\nodeJS\\documents'	'file'	'.png'	346672

Si un element no té extensió considerem que és un subdirectori.

Per obtenir el directori actual del fitxer que s'està executant podem usar la variable global de Node.js: «__dirname».

La funció «readdir()» de fs et pot ser útil, retorna un array amb el nom dels elements continguts al directori especificat.

BIBLIOGRAFIA I WEBGRAFIA

«Tutoriales Ya». https://www.tutorialesprogramacionya.com/javascriptya/nodejsya/
«w3Schools». https://www.w3schools.com/nodejs/default.asp