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 de manteniment.

Definim la funció «saludar» al fitxer «funcions.js» i fem una exportació simple perquè pugui ser importada 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 nadius interessants

- **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.

  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);
```

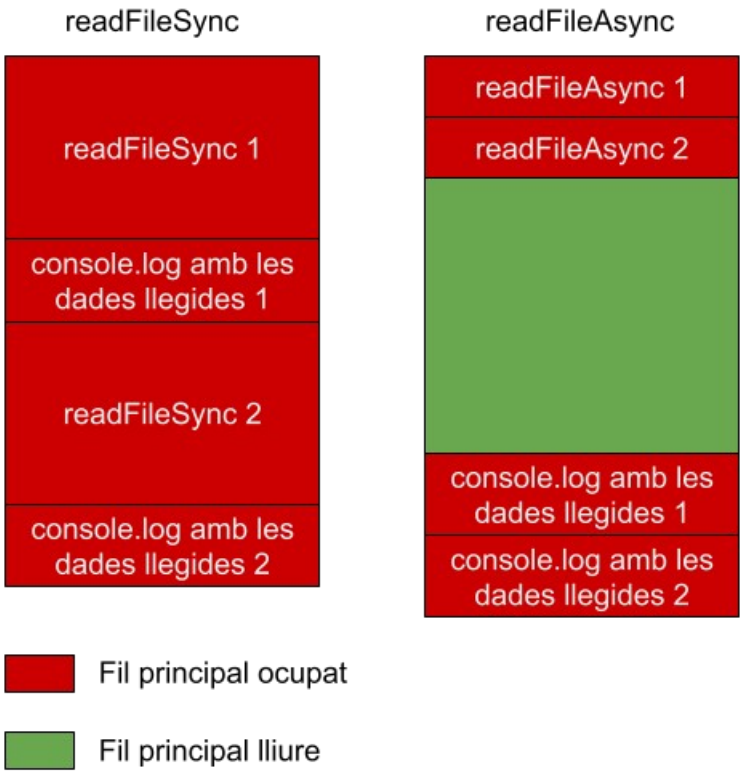
Documentació amb tots els mètodes de l'objecte EventMitter:
https://nodejs.org/api/events.html#class-eventemitter

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

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