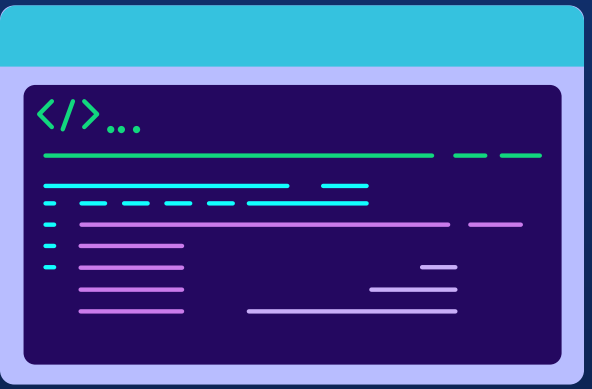
Fonctionnalités



Fonctionnalités de base nodeJS
Modules
streams and events
Implémentation d'une app HTTP

Modules de base

https://nodejs.org/api/fs.html

fs - Quelque fonctions

fs.readFile(path, callback)	Lire le contenu d'un fichier
fs.writeFile(path, data, [options])	Écrire de la donnée dans un fichier
fs.readDir(path, [options])	Afficher les fichiers de la directory
fs.mkdir(path, [options])	Créer un dossier
fs.cp(src, dest, [options])	Copier un fichier
fs.stat(path, [options])	Obtenir des informations sur un objet
fs.watch(filename, [options])	Observer les changements sur un fichier

fs - pratique

Créer un programme qui, chaque fois qu’on modifie un fichier, le remet dans son état original.

os - fonctions

os.tmpdir()	Chemin du fichier temporaire de l'OS
os.hostname()	Hostname configuré de l'OS
os.type()	Nom de l'OS
os.platform()	Plateforme de l'OS
os.arch()	Architecture CPU
os.release()	Numéro de release de l'OS
os.uptime()	uptime du système
os.totalmem()	Mémoire totale du système
os.freemem()	Mémoire vide du systeme
os.cpus()	Informations sur les CPU du systeme
os.nertworkInterfaces()	Liste des interfaces réseau

url

Le module url contient différentes fonctions permettant de traiter efficacement des URL

url - fonctions

new URL(string)	Créer un objet de type URL
url.hash	Hash a la fin de l'URL (ex: ancres de scroll)
url.host	Nom de l'host (incluant le port)
url.hostname	Nom de l'host (excluant le port)
url.href	URL entier sérialisé
url.origin	nom de domaine + protocole
url.password	mot de passe inclus dans l'URL
url.pathname	chemin après le hostname
url.port	port dans l'url. Peut être déduit du protocole
url.protocol	Portion du protocole de l'URL

https://nodejs.org/api/url.html

http

Le module HTTP permet de créer un serveur HTTP qui peut recevoir des requêtes et y répondre.

http - fonctions

https://nodejs.org/api/http.html

http.createServer((req, res) = > {})	Créer un serveur HTTP. req et res serviront à configurer le serveur.
server.listen(port)	Lancer un serveur pour qu'il écoute sur le port spécifié
req.url	URL de la requête reçue
res.writeHead(status, header)	Ajouter un header à une réponse
res.write(content)	Écrire le contenu de la réponse
res.end()	Envoyer la réponse à la requête
http.request(url, [options], [callback])	Émettre une requête HTTP

Exemple de traitement de requêtes dans un serveur avec HTTP:

```
const requestListener = function (req, res) {
  res.setHeader("Content-Type", "application/json");
  switch (req.url) {
    case "/books":
      res.writeHead(200);
      res.end(books);
      break
    case "/authors":
      res.writeHead(200);
      res.end(authors);
      break
  }
}
```

Pratique

Faire un serveur HTTP qui:

Renvoie l’heure quand on fait un GET HTTP sur /time

Renvoie la date quand on fait un GET HTTP sur /date

Événements

Les events Emitters

Beaucoup d’objets en NodeJS sont des émetteurs d’ événements.

- *fs.readStream*
- *http server*

Les events Emitters - méthodes

```
import { EventEmitter } from 'events';

const emitter = new EventEmitter();

emitter.on('messageLogged', (arg: any) => {
    console.log('Listener called', arg);
}); // listener


emitter.emit('messageLogged', { id: 1, url: 'http://'
});
```

Les events Emitters - méthodes

addListener(event, listener) on(event, listener)	Ajoute un listener à un émetteur et retourne le même émetteur
setMaxListeners(n)	Indique le nombre max de listeners que l'on peut ajouter à un émetteur
once(event, listener)	ajoute un listener qui ne sera lancé qu'une fois à un émetteur
removeAllListeners([event])	Retire tous les listeners, ou tous ceux liés à un event
listeners(event)	retourne tous les listeners pour un événement en particulier
emit(event, [arg1], [arg2], [...])	Exécute tous les listeners dans l'ordre des args donnés. Retourne vrai si l'événement avait des listeners

Les events Emitters

exemples d'événements :

error -> en cas d’erreur

newListener -> en cas de nouveau listener

removeListener -> en cas de retrait de listener

Pratique

Faire un programme qui watch les modifications d’un fichier, et écrit des logs sur un autre fichier, indiquant l’heure et la différence dans le nombre de caractères avant et après la modification.

Vous devrez utiliser les events emitters et listeners.

Streams

Les types de streams

Readable -> Pour lire de la donnée

Writable -> pour écrire de la donnée

Duplex -> Pour les deux

Transform -> pour calculer la sortie en fonction de l’entrée

Les streams sont des émetteurs d'évènements et émettent différents évènements à différents moments.

data -> événement lancé quand il reste de la data à lire

end -> événement lancé quand il n'y a plus de data à lire

error -> évènement lancé quand une erreur s'est produite durant l'IO

finish -> événement lancé quand la donnée a été vidée

Exemple: lire depuis un stream :

```
var data = "";
var readerStream = fs.createReadStream('input.txt');
readerStream.setEncoding('UTF8');

readerStream.on('data', function(chunk) {
  data += chunk;
});

readerStream.on('end',function() {
  console.log(data);
});

readerStream.on('error', function(err) {
  console.log(err.stack);
});
```

Exemple : écrire via un stream

```
var fs = require("fs");
var data = 'Simply Easy Learning';
var writerStream = fs.createWriteStream('output.txt');

writerStream.write(data,'UTF8');
writerStream.end();

writerStream.on('finish', function() {
  console.log("Write completed.");
});

writerStream.on('error', function(err) {
  console.log(err.stack);
});

console.log("Program Ended");
```

Exemple : chaîner des streams

```
var fs = require("fs");
var zlib = require('zlib');

// Compress the file input.txt to input.txt.gz
fs.createReadStream('input.txt')
  .pipe(zlib.createGzip())
  .pipe(fs.createWriteStream('input.txt.gz'));
 console.log("File Compressed.");
```

Exemple : Utiliser les streams dans les requêtes HTTP

```
const server = createServer((req, res) => { // Renvoyer a une requête POST son payload
    switch (req.method) {
        case 'POST':
            let payload = "";

            req.on('data', (chunk) => {
                payload += chunk;
            });
            req.once('end', () => {
                res.writeHead(200, { 'Content-Type': 'application/json' });
                res.end(payload);
            });
            break;
    }
});
```

Pratique

Créer un programme qui ouvre un stream d’écriture vers un fichier et fait un résumé de l'état du système toutes les secondes dans ce fichier

Pratique

écrire une fonction qui reçoit un stream et ressort le nombre de byte dans ce stream.

tester la fonction sur un fichier binaire(audio, compressé...)

tip : le type Buffer (type de chunk reçu dans le callback de `on` ou de `addListener()` a une propriete “length”

Pratique

Avec les streams, compter le nombre de mots dans un fichier texte

Pratique

Faire un serveur HTTP qui:

Quand il reçoit des requêtes les écrit dans un fichier de logs dans le dossier temporaire de l'OS sur lequel il tourne

Permet (en stockant dans la ram, pas de BDD) de créer une to do list avec des items.

les chemins seront les suivants :

POST /todolist

GET /todolists

GET /todolists/:index

DELETE /todolist/:index