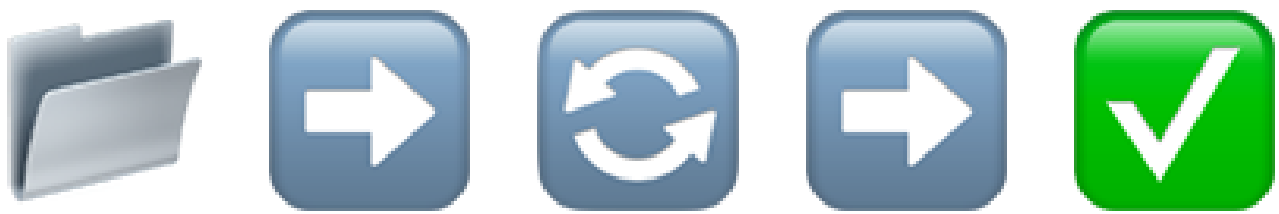
🚀 L'Asynchrone en Node.js
BUT2 - S3 INFO

Callbacks, Promises et Async/Await

Avec des exemples pratiques du système de fichiers (fs)



Joseph Azar

Maître de Conférences - Université Marie Louis Pasteur

Chercheur FEMTO-ST

🤔 Le Problème : Pourquoi l'asynchrone ?

Imagine que tu es dans un restaurant...

🚫 Mode Synchrone (Bloquant)

1. Le serveur prend ta commande
2. Il attend à ta table que le chef cuisine
3. Il t'apporte le plat
4. Il va à la table suivante

Résultat : Un client servi toutes les 30 minutes! 😱

✅ Mode Asynchrone (Non-bloquant)

1. Le serveur prend ta commande
2. Il la donne au chef
3. Il va prendre d'autres commandes
4. Quand c'est prêt, il t'apporte le plat

Résultat : Plusieurs clients servis en même temps! 🎉

🔧 Notre Exemple : Lire un fichier

On va créer un fichier texte et le lire de différentes façons

```
// D'abord, on crée notre fichier de test
const fs = require('fs');

// Créons un fichier notes.txt avec du contenu
fs.writeFileSync('notes.txt', 'Hello! Je suis le contenu du fichier.');

console.log('✅ Fichier créé!');
```

> 📝 **Ce fichier contient :** "Hello! Je suis le contenu du fichier."

1 Méthode Synchrone (Bloquante)

```
const fs = require('fs');

console.log('1. Je vais lire le fichier...');

// readFileSync = BLOQUE le programme jusqu'à ce que le fichier soit lu
const contenu = fs.readFileSync('notes.txt', 'utf-8');

console.log('2. Contenu du fichier:', contenu);
console.log('3. Fin du programme');
```

Sortie :

```
1. Je vais lire le fichier...
2. Contenu du fichier: Hello! Je suis le contenu du fichier.
3. Fin du programme
```

1 Méthode Synchrone (Bloquante)

```
const fs = require('fs');

console.log('1. Je vais lire le fichier...');

// readFileSync = BLOQUE le programme jusqu'à ce que le fichier soit lu
const contenu = fs.readFileSync('notes.txt', 'utf-8');

console.log('2. Contenu du fichier:', contenu);
console.log('3. Fin du programme');
```

Sortie :

```
1. Je vais lire le fichier...
2. Contenu du fichier: Hello! Je suis le contenu du fichier.
3. Fin du programme
```

⚠️ **Problème :** Si le fichier est gros (1GB), ton programme est BLOQUÉ pendant toute la lecture!

😰 Pourquoi le Synchrone c'est MAL ?

```
const fs = require('fs');

console.log('🏁 Début');

// Imagine que ce fichier fait 1GB
const fichier1 = fs.readFileSync('gros-fichier-1.txt', 'utf-8');  // ⏱️ 5 secondes
console.log('✅ Fichier 1 lu');

const fichier2 = fs.readFileSync('gros-fichier-2.txt', 'utf-8');  // ⏱️ 5 secondes
console.log('✅ Fichier 2 lu');

const fichier3 = fs.readFileSync('gros-fichier-3.txt', 'utf-8');  // ⏱️ 5 secondes
console.log('✅ Fichier 3 lu');

console.log('🏁 Fin');

// TOTAL : 15 secondes ! 😭
```

❌ **Résultat :** 15 secondes d'attente, CPU utilisé à 0%, programme figé!

🔄 Solution 1 : Les Callbacks

La première façon de gérer l'asynchrone

🎯 C'est quoi un Callback ?

Un callback c'est une fonction qu'on donne à une autre fonction pour qu'elle l'appelle plus tard

> **Analogie :** Tu donnes ton numéro de téléphone au restaurant. Ils t'appellent quand c'est prêt!

```
// Syntaxe générale d'un callback
fonction(parametres, (error, resultat) => {
    // Cette fonction sera appelée PLUS TARD
    if (error) {
        // Gérer l'erreur
    } else {
        // Utiliser le résultat
    }
});
```

📖 Lire un fichier avec Callback

```
const fs = require('fs');

console.log('1. Je lance la lecture...');

fs.readFile('notes.txt', 'utf-8', (error, data) => {
    // Cette fonction sera appelée QUAND la lecture sera finie
    if (error) {
        console.log('❌ Erreur:', error);
    } else {
        console.log('3. Contenu:', data);
    }
});

console.log('2. Je continue mon programme!');
```

Sortie (ATTENTION à l'ordre!) :

```
1. Je lance la lecture...
2. Je continue mon programme!
3. Contenu: Hello! Je suis le contenu du fichier.
```

📖 Lire un fichier avec Callback

```
const fs = require('fs');

console.log('1. Je lance la lecture...');

fs.readFile('notes.txt', 'utf-8', (error, data) => {
    // Cette fonction sera appelée QUAND la lecture sera finie
    if (error) {
        console.log('❌ Erreur:', error);
    } else {
        console.log('3. Contenu:', data);
    }
});

console.log('2. Je continue mon programme!');
```

Sortie (ATTENTION à l'ordre!) :

```
1. Je lance la lecture...
2. Je continue mon programme!
3. Contenu: Hello! Je suis le contenu du fichier.
```

📚 Lire plusieurs fichiers avec Callbacks

```
const fs = require('fs');

// Créons 3 fichiers
fs.writeFileSync('file1.txt', 'Contenu 1');
fs.writeFileSync('file2.txt', 'Contenu 2');
fs.writeFileSync('file3.txt', 'Contenu 3');

// Lire les 3 fichiers EN PARALLÈLE
console.log('🏁 Début des lectures');

fs.readFile('file1.txt', 'utf-8', (err, data) => {
    console.log('📄 File 1:', data);
});

fs.readFile('file2.txt', 'utf-8', (err, data) => {
    console.log('📄 File 2:', data);
});

fs.readFile('file3.txt', 'utf-8', (err, data) => {
    console.log('📄 File 3:', data);
});

console.log('🚀 Toutes les lectures sont lancées!');

// Les 3 fichiers sont lus EN MÊME TEMPS! 🎉
```

😈 Le Problème : Callback Hell

Que faire si on veut lire les fichiers dans l'ordre ?

```
// ❌ MAUVAIS : Le fameux "Callback Hell"
fs.readFile('file1.txt', 'utf-8', (err1, data1) => {
    if (err1) {
        console.log('Erreur file1');
    } else {
        console.log('File 1:', data1);
        fs.readFile('file2.txt', 'utf-8', (err2, data2) => {
            if (err2) {
                console.log('Erreur file2');
            } else {
                console.log('File 2:', data2);
                fs.readFile('file3.txt', 'utf-8', (err3, data3) => {
                    if (err3) {
                        console.log('Erreur file3');
                    } else {
                        console.log('File 3:', data3);
                        // 😱 Imagine avec 10 fichiers...
                    }
                });
            }
        });
    }
});
```

Solution 2 : Les Promises

Une meilleure façon de gérer l'asynchrone

🤝 C'est quoi une Promise ?

> **Analogie :** Une promesse c'est comme un ticket de commande au restaurant
>
> - 📝 **Pending** : En attente (la commande est en cours)
> - ✅ **Fulfilled** : Réussie (voici ton plat!)
> - ❌ **Rejected** : Échouée (désolé, plus de pizza!)

```
// Une Promise a deux méthodes importantes :
promise
    .then(resultat => {
        // Si tout va bien
    })
    .catch(erreur => {
        // Si ça plante
    });
```

🔄 Transformer un Callback en Promise

```
const fs = require('fs');
const { promisify } = require('util');

// Méthode 1 : Utiliser promisify (FACILE!)
const readFilePromise = promisify(fs.readFile);

// Méthode 2 : Créer sa propre Promise (pour comprendre)
function lireFichier(nomFichier) {
    return new Promise((resolve, reject) => {
        fs.readFile(nomFichier, 'utf-8', (error, data) => {
            if (error) {
                reject(error);  // Promise échouée
            } else {
                resolve(data);  // Promise réussie
            }
        });
    });
}
```

✨ Utiliser les Promises

```
const fs = require('fs');
const { promisify } = require('util');
const readFilePromise = promisify(fs.readFile);

console.log('1. Je lance la lecture...');

readFilePromise('notes.txt', 'utf-8')
    .then(data => {
        console.log('3. Contenu:', data);
        return data.toUpperCase(); // On peut retourner une valeur
    })
    .then(dataEnMajuscules => {
        console.log('4. En majuscules:', dataEnMajuscules);
    })
    .catch(error => {
        console.log('❌ Erreur:', error);
    });

console.log('2. Je continue!');
```

Sortie :

```
1. Je lance la lecture...
2. Je continue!
```

⛓️ Chaîner les Promises (Bye Bye Callback Hell!)

```
const fs = require('fs');
const { promisify } = require('util');
const readFilePromise = promisify(fs.readFile);

// Lire 3 fichiers dans l'ordre SANS callback hell!
readFilePromise('file1.txt', 'utf-8')
    .then(data1 => {
        console.log('File 1:', data1);
        return readFilePromise('file2.txt', 'utf-8');
    })
    .then(data2 => {
        console.log('File 2:', data2);
        return readFilePromise('file3.txt', 'utf-8');
    })
    .then(data3 => {
        console.log('File 3:', data3);
        console.log('✅ Tous les fichiers lus!');
    })
    .catch(error => {
        console.log('❌ Erreur:', error);
    });

// BEAUCOUP plus lisible! 🎉
```

🚀 Promise.all - Lire en parallèle

```
const fs = require('fs');
const { promisify } = require('util');
const readFilePromise = promisify(fs.readFile);

// Lire TOUS les fichiers EN MÊME TEMPS
const promesses = [
    readFilePromise('file1.txt', 'utf-8'),
    readFilePromise('file2.txt', 'utf-8'),
    readFilePromise('file3.txt', 'utf-8')
];

Promise.all(promesses)
    .then(resultats => {
        // resultats est un tableau avec TOUS les contenus
        console.log('File 1:', resultats[0]);
        console.log('File 2:', resultats[1]);
        console.log('File 3:', resultats[2]);
        console.log('✅ Tout est lu EN PARALLÈLE!');
    })
    .catch(error => {
        console.log('❌ Une des lectures a échoué:', error);
    });

// Super rapide car tout est fait en même temps! ⚡
```

🌟 Solution 3 : Async/Await

La façon MODERNE (et la plus simple!)

😍 Async/Await : Le Code Synchrone qui est Asynchrone!

C'est quoi ?

- `async` : Dit qu'une fonction est asynchrone
- `await` : Attend le résultat d'une Promise
- Ça ressemble à du code synchrone mais c'est asynchrone!

```
// Règles importantes :
// 1. await ne fonctionne QUE dans une fonction async
// 2. await attend une Promise
// 3. try/catch pour gérer les erreurs

async function maFonction() {
    try {
        const resultat = await unePromise();
        console.log(resultat);
    } catch (error) {
        console.log('Erreur:', error);
    }
```

📖 Lire un fichier avec Async/Await

```
const fs = require('fs');
const { promisify } = require('util');
const readFilePromise = promisify(fs.readFile);

// IMPORTANT : La fonction DOIT être async
async function lireFichier() {
    console.log('1. Je vais lire le fichier...');

    try {
        // await = "attends que ce soit fini"
        const contenu = await readFilePromise('notes.txt', 'utf-8');
        console.log('2. Contenu:', contenu);

        // On peut faire plusieurs await
        const contenuMajuscules = contenu.toUpperCase();
        console.log('3. En majuscules:', contenuMajuscules);

    } catch (error) {
        console.log('❌ Erreur:', error);
    }

    console.log('4. Fin de la lecture!');
}

// N'oublie pas d'appeler la fonction!
lireFichier();
```

Plusieurs fichiers avec Async/Await

Dans l'ordre (séquentiel)

```
async function lireDansLOrdre() {
    try {
        const data1 = await readFilePromise('file1.txt', 'utf-8')
        console.log('File 1:', data1);

        const data2 = await readFilePromise('file2.txt', 'utf-8')
        console.log('File 2:', data2);

        const data3 = await readFilePromise('file3.txt', 'utf-8')
        console.log('File 3:', data3);

    } catch (error) {
        console.log('Erreur:', error);
    }
}
```

🎯 Exemple Complet : Les 4 Approches

On va écrire puis lire un fichier avec chaque méthode

```
const fs = require('fs');
const { promisify } = require('util');

// 1. SYNCHRONE (Bloquant)
function methodeSync() {
    fs.writeFileSync('test.txt', 'Hello World!');
    const contenu = fs.readFileSync('test.txt', 'utf-8');
    console.log('Sync:', contenu);
}

// 2. CALLBACK
function methodeCallback() {
    fs.writeFile('test.txt', 'Hello World!', (err) => {
        if (err) throw err;
        fs.readFile('test.txt', 'utf-8', (err, data) => {
            if (err) throw err;
            console.log('Callback:', data);
        });
    });
```

Exemple du Monde Réel : Notes App

Transformons le module notes.js en version moderne!

```
const fs = require('fs').promises;  // Version Promise de fs

// Charger les notes - Version CALLBACK originale
const loadNotesCallback = (callback) => {
    fs.readFile('notes.json', (err, dataBuffer) => {
        if (err) {
            callback(err, []);
        } else {
            const dataJSON = dataBuffer.toString();
            const notes = JSON.parse(dataJSON);
            callback(null, notes);
        }
    });
};

// Charger les notes - Version PROMISE
const loadNotesPromise = () => {
    return fs.readFile('notes.json')
        .then(dataBuffer => {
            const dataJSON = dataBuffer.toString();
```

🎮 Utiliser nos 3 versions

```
// UTILISATION DES 3 VERSIONS

// 1. CALLBACK
loadNotesCallback((error, data) => {
    if (error) {
        console.log('Erreur:', error);
    } else {
        console.log('Notes (callback):', data);
    }
});

// 2. PROMISE
loadNotesPromise()
    .then(data => console.log('Notes (promise):', data))
    .catch(error => console.log('Erreur:', error));

// 3. ASYNC/AWAIT
const afficherNotes = async () => {
    try {
        const mesNotes = await loadNotesAsync();
        console.log('Notes (async/await):', mesNotes);
    } catch(error) {
        console.log('Erreur:', error);
    }
};

afficherNotes();
```

🏋️ Exercice 1 : Copier un fichier

Écris une fonction qui copie un fichier de façon asynchrone

> **Mission :** Lire source.txt et écrire son contenu dans destination.txt

```
const fs = require('fs').promises;

// À TOI DE JOUER!
async function copierFichier(source, destination) {
    // 1. Lire le fichier source
    // 2. Écrire dans le fichier destination
    // 3. Afficher "Copie terminée!"
    // 4. Gérer les erreurs
}

// Test
copierFichier('source.txt', 'destination.txt');
```

🏋️ Exercice 1 : Copier un fichier

Écris une fonction qui copie un fichier de façon asynchrone

> **Mission :** Lire source.txt et écrire son contenu dans destination.txt

```
const fs = require('fs').promises;

// À TOI DE JOUER!
async function copierFichier(source, destination) {
    // 1. Lire le fichier source
    // 2. Écrire dans le fichier destination
    // 3. Afficher "Copie terminée!"
    // 4. Gérer les erreurs
}

// Test
copierFichier('source.txt', 'destination.txt');
```

Voir la solution

✅ Solution Exercice 1

```
const fs = require('fs').promises;

async function copierFichier(source, destination) {
    try {
        // 1. Lire le fichier source
        console.log(`📖 Lecture de ${source}...`);
        const contenu = await fs.readFile(source, 'utf-8');

        // 2. Écrire dans le fichier destination
        console.log(`✏️ Écriture dans ${destination}...`);
        await fs.writeFile(destination, contenu);

        // 3. Confirmer
        console.log('✅ Copie terminée!');

    } catch (error) {
        console.log('❌ Erreur lors de la copie:', error.message);
    }
}

// Test
copierFichier('source.txt', 'destination.txt');
```

🏋️ Exercice 2 : Lire plusieurs fichiers

Lis 3 fichiers et combine leur contenu

```
const fs = require('fs').promises;

// Crée d'abord les fichiers de test
async function creerFichiers() {
    await fs.writeFile('part1.txt', 'Début de l\'histoire. ');
    await fs.writeFile('part2.txt', 'Milieu de l\'histoire. ');
    await fs.writeFile('part3.txt', 'Fin de l\'histoire.');
}

// À TOI DE JOUER!
async function combinerFichiers() {
    // 1. Lire les 3 fichiers EN PARALLÈLE
    // 2. Combiner leur contenu
    // 3. Écrire dans histoire-complete.txt
    // 4. Afficher le résultat
}

// Test
creerFichiers().then(() => combinerFichiers());
```

🏋️ Exercice 2 : Lire plusieurs fichiers

Lis 3 fichiers et combine leur contenu

```
const fs = require('fs').promises;

// Crée d'abord les fichiers de test
async function creerFichiers() {
    await fs.writeFile('part1.txt', 'Début de l\'histoire. ');
    await fs.writeFile('part2.txt', 'Milieu de l\'histoire. ');
    await fs.writeFile('part3.txt', 'Fin de l\'histoire.');
}

// À TOI DE JOUER!
async function combinerFichiers() {
    // 1. Lire les 3 fichiers EN PARALLÈLE
    // 2. Combiner leur contenu
    // 3. Écrire dans histoire-complete.txt
    // 4. Afficher le résultat
}

// Test
creerFichiers().then(() => combinerFichiers());
```

Voir la solution

✅ Solution Exercice 2

```
const fs = require('fs').promises;

async function combinerFichiers() {
    try {
        console.log('📚 Lecture des 3 parties...');

        // Lire les 3 fichiers EN PARALLÈLE (plus rapide!)
        const [part1, part2, part3] = await Promise.all([
            fs.readFile('part1.txt', 'utf-8'),
            fs.readFile('part2.txt', 'utf-8'),
            fs.readFile('part3.txt', 'utf-8')
        ]);

        // Combiner
        const histoireComplete = part1 + part2 + part3;

        // Écrire le résultat
        await fs.writeFile('histoire-complete.txt', histoireComplete);

        console.log('✅ Histoire complète:');
        console.log(histoireComplete);

    } catch (error) {
        console.log('❌ Erreur:', error.message);
    }
}
```

⚠️ Erreur Courante #1 : Oublier async

❌ NE MARCHE PAS - await sans async

```
// ❌ ERREUR : SyntaxError
function maFonction() {
    const data = await fs.readFile('file.txt');
    // SyntaxError: Unexpected identifier
    // await is only valid in async functions
}
```

✅ CORRECT - Ajouter le mot-clé async

```
// ✅ BON
async function maFonction() {
    const data = await fs.readFile('file.txt');
    // Maintenant ça marche!
}
```

⚠️ Erreur Courante #1 : Oublier async

❌ NE MARCHE PAS - await sans async

```
// ❌ ERREUR : SyntaxError
function maFonction() {
    const data = await fs.readFile('file.txt');
    // SyntaxError: Unexpected identifier
    // await is only valid in async functions
}
```

✅ CORRECT - Ajouter le mot-clé async

```
// ✅ BON
async function maFonction() {
    const data = await fs.readFile('file.txt');
    // Maintenant ça marche!
}
```

📝 **Règle à retenir :** Si tu utilises `await` dans une fonction, elle DOIT être `async`

⚠️ Erreur Courante #2 : Oublier await

❌ NE MARCHE PAS - Promise au lieu du résultat

```
// ❌ ERREUR : On récupère une Promise, pas le contenu
async function maFonction() {
    const data = fs.readFile('file.txt', 'utf-8');
    console.log(data);
    // Affiche: Promise { <pending> }
    // PAS le contenu du fichier!
}
          </pending>
```

✅ CORRECT - Ajouter await pour attendre le résultat

```
// ✅ BON
async function maFonction() {
    const data = await fs.readFile('file.txt', 'utf-8');
    console.log(data);
    // Affiche: Le contenu du fichier!
}
```

⚠️ Erreur Courante #2 : Oublier await

❌ NE MARCHE PAS - Promise au lieu du résultat

```
// ❌ ERREUR : On récupère une Promise, pas le contenu
async function maFonction() {
    const data = fs.readFile('file.txt', 'utf-8');
    console.log(data);
    // Affiche: Promise { <pending> }
    // PAS le contenu du fichier!
}
          </pending>
```

✅ CORRECT - Ajouter await pour attendre le résultat

```
// ✅ BON
async function maFonction() {
    const data = await fs.readFile('file.txt', 'utf-8');
    console.log(data);
    // Affiche: Le contenu du fichier!
}
```

📝 **Règle à retenir :** Sans `await`, tu récupères une

Gestion d'Erreurs : Best Practices

```
const fs = require('fs').promises;

// ✅ BONNE PRATIQUE : Toujours gérer les erreurs!
async function lireFichierSecurise(nomFichier) {
    try {
        // Vérifier que le fichier existe
        await fs.access(nomFichier);

        // Lire le fichier
        const contenu = await fs.readFile(nomFichier, 'utf-8');

        // Vérifier que ce n'est pas vide
        if (!contenu || contenu.trim() === '') {
            throw new Error('Le fichier est vide!');
        }

        return contenu;

    } catch (error) {
        // Gérer différents types d'erreurs
        if (error.code === 'ENOENT') {
            console.log('❌ Le fichier n\'existe pas!');
        } else if (error.code === 'EACCES') {
            console.log('❌ Pas de permission de lecture!');
        } else {
            console.log('❌ Erreur:', error.message);
        }
```

⚡ Comparaison des Performances

```
const fs = require('fs').promises;

// Test : Lire 100 petits fichiers

// 🐌 LENT : Séquentiel
async function lentSequentiel(files) {
    console.time('Séquentiel');
    for (const file of files) {
        await fs.readFile(file);
    }
    console.timeEnd('Séquentiel'); // ~500ms
}

// 🚀 RAPIDE : Parallèle
async function rapideParallele(files) {
    console.time('Parallèle');
    await Promise.all(
        files.map(file => fs.readFile(file))
    );
    console.timeEnd('Parallèle'); // ~100ms
}

// 5x plus rapide en parallèle! 🎉
```

💡 **Règle d'or :** Utilise Promise.all() quand tu peux faire

🤔 Quand Utiliser Quoi ?

Méthode	Quand l'utiliser	Avantages	Inconvénients
Synchrone	Scripts simples, config au démarrage	Simple, direct	Bloque tout
Callbacks	Vieux code, libs anciennes	Compatible partout	Callback hell
Promises	Chaîner des opérations	Chainable, propre	Syntaxe .then()
	👑		

📁 Structure d'un Vrai Projet

```
// utils/fileHelpers.js
const fs = require('fs').promises;
const path = require('path');

class FileManager {
    async readJSON(filePath) {
        try {
            const data = await fs.readFile(filePath, 'utf-8');
            return JSON.parse(data);
        } catch (error) {
            if (error.code === 'ENOENT') {
                return null;  // Fichier n'existe pas
            }
            throw error;  // Autre erreur
        }
    }

    async writeJSON(filePath, data) {
        const dir = path.dirname(filePath);
        await fs.mkdir(dir, { recursive: true });  // Créer le dossier si nécessaire
        await fs.writeFile(filePath, JSON.stringify(data, null, 2));
    }

    async exists(filePath) {
        try {
            await fs.access(filePath);
            return true;
```

🎯 Utiliser notre FileManager

```
// app.js
const fileManager = require('./utils/fileHelpers');

async function main() {
    // Charger la config
    const config = await fileManager.readJSON('config.json');
    if (!config) {
        console.log('Première exécution, création de la config...');
        await fileManager.writeJSON('config.json', {
            appName: 'Super App',
            version: '1.0.0'
        });
    }

    // Charger les données utilisateur
    const userData = await fileManager.readJSON('data/users.json') || [];

    // Ajouter un utilisateur
    userData.push({
        id: Date.now(),
        name: 'Alice',
        createdAt: new Date()
    });

    // Sauvegarder
    await fileManager.writeJSON('data/users.json', userData);
    console.log('✅ Utilisateur ajouté!');
```

🚀 Mini-Projet : ToDo List Asynchrone

Créons une app de todo list avec fs asynchrone!

```
const fs = require('fs').promises;

class TodoApp {
    constructor(filePath = 'todos.json') {
        this.filePath = filePath;
    }

    async loadTodos() {
        try {
            const data = await fs.readFile(this.filePath, 'utf-8');
            return JSON.parse(data);
        } catch {
            return [];
        }
    }

    async saveTodos(todos) {
        await fs.writeFile(this.filePath, JSON.stringify(todos, null, 2));
    }
```

📝 Récapitulatif

Ce qu'on a appris

- ✅ Différence sync/async
- ✅ Callbacks et leurs problèmes
- ✅ Promises et chaînage
- ✅ Async/Await moderne
- ✅ Gestion d'erreurs
- ✅ Lecture/écriture de fichiers
- ✅ Opérations parallèles

À retenir

🌟 **Utilise Async/Await!**

- Plus lisible
- Plus simple
- Moins d'erreurs
- Standard moderne

📚 Pour Aller Plus Loin

Documentation

- 📖 Node.js fs.promises : nodejs.org/api/fs
- 📖 MDN Async/Await : MDN Web Docs
- 📖 JavaScript.info : javascript.info/async

Exercices Supplémentaires

1. Créer un système de backup qui copie tous les .txt d'un dossier
2. Lire un CSV et le convertir en JSON
3. Créer un watcher qui surveille les changements dans un fichier
4. Implémenter un système de cache avec expiration

💡 Conseils Finaux

Les Commandements de l'Asynchrone

1. 🌟 **Tu utiliseras Async/Await** pour du nouveau code
2. ⚡ **Tu paralléliseras** avec Promise.all() quand possible
3. 🛡️ **Tu géreras les erreurs** avec try/catch
4. 📝 **Tu n'oublieras pas** le mot-clé await
5. 🚫 **Tu éviteras** le callback hell
6. ✅ **Tu testeras** ton code asynchrone

🎉 Félicitations!

Tu maîtrises maintenant l'asynchrone en Node.js!

Va coder des trucs géniaux! 🚀

? Questions ?

N'hésitez pas à poser vos questions!



Joseph Azar

Maître de Conférences - Université Marie Louis Pasteur

Chercheur FEMTO-ST

joseph.azar@univ-fcomte.fr