Cours Pratique: Développement Backend avec Node.js - Module 1

Module 1 : Introduction à Node.js (3 heures)

Introduction à Node.js

Objectif : Comprendre les bases de Node.js, son architecture et son fonctionnement.

Qu'est-ce que Node.js ?

Node.js est un environnement d'exécution JavaScript qui permet d'exécuter du code JavaScript en dehors d'un navigateur web. Il est basé sur le moteur V8 de Google Chrome, ce qui signifie qu'il utilise le même moteur que celui utilisé par Chrome pour interpréter le code JavaScript. Node.js est conçu pour développer des applications réseau rapides et évolutives.

Pourquoi choisir Node.js ?

• Asynchrone et non-bloquant : Node.js utilise une architecture basée sur les événements, ce qui signifie qu'il peut gérer de nombreuses requêtes simultanément sans être bloqué.

• Haute performance : Le moteur V8 de Google Chrome est très rapide, ce qui rend Node.js performant pour des applications backend.

• Large écosystème : NPM (Node Package Manager) propose plus d'un million de packages, ce qui facilite le développement de projets.

• Utilisation de JavaScript : Les développeurs peuvent utiliser la même langue de programmation pour le frontend et le backend.

1. Asynchrone : Qu'est-ce que cela signifie ?

En programmation, l'exécution asynchrone signifie qu'une opération peut être lancée sans attendre qu'elle se termine pour passer à l'instruction suivante. Cela permet au programme de continuer à s'exécuter et de ne pas être bloqué en attendant la fin de l'opération.

2. Non-bloquant : Qu'est-ce que cela signifie ?

Le terme "non-bloquant" signifie que Node.js peut lancer plusieurs opérations simultanément et ne sera pas bloqué en attendant que l'une d'entre elles se termine. Cela est particulièrement utile pour les opérations qui prennent du temps, comme la lecture de fichiers, l'accès à une base de données, ou la réalisation de requêtes réseau.

Exemple d'opération Bloquante (Synchrone) :

Voici un exemple en utilisant une fonction synchrone pour lire un fichier :

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

// Lecture synchrone d'un fichier
const data = fs.readFileSync('example.txt', 'utf8');
console.log(data);

console.log('Ceci s\'affichera après la lecture du fichier.');
```

Explication :

- fs.readFileSync est une fonction synchrone qui lit le contenu du fichier.
- Le programme est bloqué jusqu'à ce que le fichier soit complètement lu.
- La ligne console.log('Ceci s\'affichera après la lecture du fichier.'); ne sera exécutée qu'après la lecture complète du fichier.

Exemple d'opération Asynchrone et Non-bloquante :

Voici le même exemple en utilisant la fonction asynchrone fs.readFile :

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

// Lecture asynchrone d'un fichier
fs.readFile('example.txt', 'utf8', (err, data) => {
 if (err) {
  console.error(err);
  return;
 }
 console.log(data);
});

console.log('Ceci s\'affiche avant la lecture du fichier.');
```

Explication :

- fs.readFile est une fonction asynchrone. Elle démarre la lecture du fichier, puis passe immédiatement à l'instruction suivante.
- Le programme n'est pas bloqué, et console.log('Ceci s\'affiche avant la lecture du fichier.'); s'exécute avant même que le contenu du fichier ne soit affiché.
- Une fois la lecture terminée, la fonction de rappel (err, data) => { ... } est exécutée pour afficher le contenu.

Avantages de l'asynchrone et du non-bloquant :

1. **Performances améliorées** : Les opérations lourdes n'arrêtent pas le programme, permettant à d'autres tâches d'être exécutées.
2. **Gestion efficace des ressources** : Node.js peut gérer de nombreuses connexions simultanément, ce qui en fait un excellent choix pour les applications en temps réel (comme les serveurs web).

Exemple réel : Serveur HTTP avec Node.js

Créons un serveur HTTP simple qui simule une opération lente (comme une requête à une base de données) :

Code Synchrone (Bloquant) :

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

const server = http.createServer((req, res) => {
  if (req.url === '/lente') {
    // Simulation d'une opération lente
    const start = Date.now();
    while (Date.now() - start < 5000) {} // Bloque pendant 5 secondes
    res.end('Opération lente terminée.');
  } else {
    res.end('Hello World');
  }
});

server.listen(3000, () => {
  console.log('Serveur en écoute sur le port 3000');
});
```

Problème : Pendant les 5 secondes d'attente, le serveur est bloqué et ne peut pas traiter d'autres requêtes.

Code Asynchrone (Non-bloquant) :

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

const server = http.createServer((req, res) => {
  if (req.url === '/lente') {
    // Simulation d'une opération lente non-bloquante
    setTimeout(() => {
      res.end('Opération lente terminée.');
    }, 5000);
  } else {
    res.end('Hello World');
  }
});
```

```
server.listen(3000, () => {
  console.log('Serveur en écoute sur le port 3000');
});
```

Explication :

- Avec setTimeout, le serveur ne bloque pas pendant les 5 secondes d'attente. Il peut continuer à traiter d'autres requêtes pendant ce temps.

Conclusion

- **Asynchrone** : Les opérations peuvent démarrer et se terminer sans attendre les unes les autres.
- **Non-bloquant** : Node.js ne s'arrête pas pour attendre la fin d'une tâche longue, ce qui permet une exécution efficace des tâches multiples.

Ce mécanisme rend Node.js très performant pour les applications en temps réel, telles que les serveurs de chat, les applications de streaming, et les API hautement interactives.

Installation de Node.js

1. Allez sur le site officiel : https://nodejs.org et téléchargez la version LTS (Long-Term Support).

2. Suivez les instructions d'installation adaptées à votre système d'exploitation.

3. Vérifiez l'installation en utilisant les commandes suivantes dans votre terminal :

- `node -v` : affiche la version de Node.js installée.

- `npm -v` : affiche la version de NPM installée.

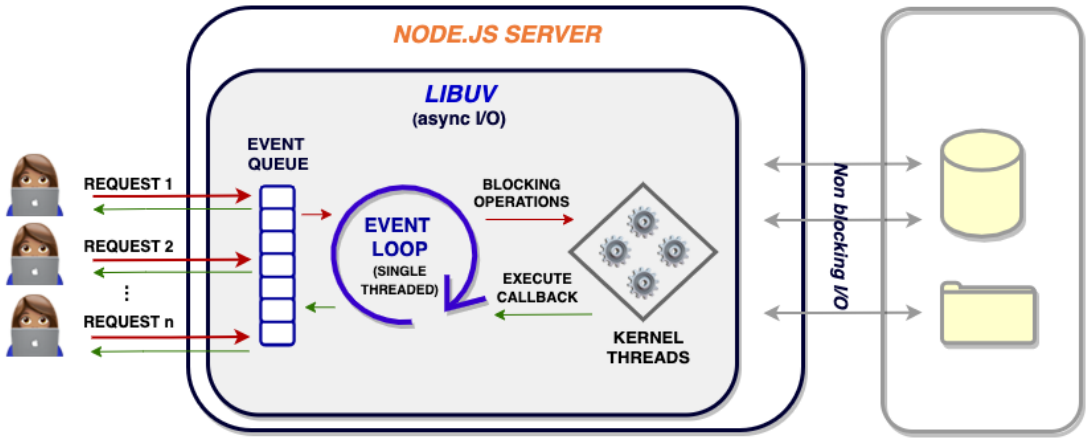
Architecture de Node.js

Node.js est basé sur une architecture événementielle non-bloquante :

• Event Loop : C'est la boucle qui permet à Node.js de gérer les opérations asynchrones.

• Non-blocking I/O : Node.js peut gérer plusieurs requêtes simultanément sans attendre que l'une d'entre elles se termine.

• Single-thread : Node.js utilise un seul thread pour gérer toutes les requêtes, ce qui le rend léger et efficace.



Votre premier programme Node.js

Créez un fichier appelé `app.js` et ajoutez le code suivant :

```javascript
console.log('Bonjour, Node.js !');
```

Ensuite, exécutez le fichier avec la commande : `node app.js`

Les modules intégrés de Node.js

Node.js propose de nombreux modules intégrés, tels que :

• `fs` : pour gérer les opérations sur les fichiers.

• `http` : pour créer des serveurs web.

• `path` : pour travailler avec les chemins de fichiers.

Exemple d'utilisation du module `os` :

```javascript
const os = require('os');
console.log('Système d'exploitation :', os.platform());
```

Utiliser NPM (Node Package Manager)

NPM est le gestionnaire de packages de Node.js qui permet d'installer des bibliothèques externes.

• Pour initialiser un projet Node.js, utilisez la commande : `npm init`

• Pour installer un package, utilisez : `npm install nom_du_package`

Exemple : `npm install lodash` pour installer la bibliothèque lodash.

Structure d'un projet Node.js

Lors de l'initialisation d'un projet Node.js, les fichiers principaux sont :

• `package.json` : contient les informations du projet et les dépendances.

• `node_modules/` : répertoire contenant les packages installés.

Exercice Pratique

Créer un programme qui affiche les informations système :

1. Initialisez un projet Node.js avec `npm init`.

2. Installez le package `os` si besoin.

3. Créez un fichier `systemInfo.js` avec le code suivant :

```javascript
const os = require('os');
console.log('Plateforme:', os.platform());
console.log('Architecture:', os.arch());
console.log('Mémoire totale:', os.totalmem());
console.log('Mémoire libre:', os.freemem());
```

4. Exécutez le programme avec `node systemInfo.js`.

Correction de l'exercice

Le programme affiche les informations du système telles que :

• Plateforme (ex: linux, win32)

• Architecture (ex: x64)

• Mémoire totale en octets

• Mémoire libre en octets

Création d'un module personnalisé

1. Créez un fichier mathOperations.js qui exporte deux fonctions :
    - add(a, b) : additionne deux nombres
    - multiply(a, b) : multiplie deux nombres
2. Créez un fichier main.js qui importe mathOperations.js et utilise les deux fonctions pour afficher les résultats de 5 + 3 et 5 * 3.

Corrigé :

mathOperations.js :

```
function add(a, b) {
  return a + b;
}

function multiply(a, b) {
  return a * b;
}

module.exports = {
  add,
  multiply
};
```

main.js :

```
const math = require('./mathOperations');

console.log('5 + 3 =', math.add(5, 3));
console.log('5 * 3 =', math.multiply(5, 3));
```

Exécutez node main.js pour voir les résultats :

```
5 + 3 = 8
5 * 3 = 15
```

Créer un petit serveur HTTP

1. Créez un fichier nommé server.js.
2. Utilisez le module intégré http pour créer un serveur web qui :
    - Répond à la route / avec le message "Bienvenue sur mon serveur Node.js !"

- Répond à la route /date avec la date et l'heure actuelles.
- Répond avec "Page non trouvée" pour toutes les autres routes.

3. Faites en sorte que le serveur écoute sur le port 3000.

```
// Contenu du fichier server.js
const http = require('http');

// Créer un serveur
const server = http.createServer((req, res) => {
  // Définir l'en-tête de la réponse
  res.writeHead(200, { 'Content-Type': 'text/plain' });

  // Routes
  if (req.url === '/') {
    res.end('Bienvenue sur mon serveur Node.js !');
  } else if (req.url === '/date') {
    const currentDate = new Date();
    res.end(`La date et l'heure actuelles sont : ${currentDate}`);
  } else {
    res.writeHead(404);
    res.end('Page non trouvée');
  }
});

// Démarrer le serveur sur le port 3000
server.listen(3000, () => {
  console.log('Serveur en écoute sur http://localhost:3000');
});
```

Comment tester :

1. Exécutez le serveur avec la commande : node server.js
2. Ouvrez votre navigateur web et accédez à l'URL suivante :
   - http://localhost:3000 ➞ Affichera "Bienvenue sur mon serveur Node.js !"
   - http://localhost:3000/date ➞ Affichera la date et l'heure actuelles
   - http://localhost:3000/anything-else ➞ Affichera "Page non trouvée"

Cet exercice vous aidera à comprendre comment créer un serveur HTTP basique avec Node.js et à gérer différentes routes, ce qui est une compétence fondamentale pour le développement backend.