Réponses à l'Examen Théorique sur Node.js

TOYI françois

November 2024

1 Introduction à Node.js

1. **Définissez ce qu'est Node.js. Précisez son architecture et son rôle principal dans le développement d'applications modernes.**
   Node.js est un environnement d'exécution JavaScript côté serveur, construit sur le moteur V8 de Google. Son architecture est non bloquante et orientée événements, ce qui le rend efficace pour des applications nécessitant une forte intensité d'entrées/sorties (E/S) et pouvant gérer de nombreuses connexions simultanées, comme les serveurs web et les APIs RESTful.

2. **Qu'est-ce que le moteur V8 ? Expliquez son importance dans le fonctionnement de Node.js.**
   Le moteur V8 est un moteur JavaScript open-source développé par Google, qui compile du code JavaScript en code machine natif, rendant ainsi l'exécution rapide et efficace. V8 est crucial pour Node.js car il permet d'exécuter le JavaScript côté serveur avec des performances élevées.

3. **Expliquez la différence entre JavaScript côté serveur et JavaScript côté client. Donnez deux exemples pour chaque cas.**
   JavaScript côté client s'exécute dans le navigateur pour gérer l'interactivité de l'interface utilisateur, par exemple, pour manipuler le DOM ou valider des formulaires. JavaScript côté serveur, comme avec Node.js, s'exécute sur le serveur pour traiter des requêtes, interagir avec la base de données, et générer des réponses HTTP.

4. **Node.js est dit "asynchrone" et "orienté événements". Expliquez ce que cela signifie en décrivant l'Event Loop.**
   L'asynchronisme de Node.js signifie qu'il ne bloque pas les opérations E/S ; au lieu de cela, il utilise des callbacks pour exécuter des tâches une fois qu'elles sont terminées. L'Event Loop gère la file d'attente des callbacks et exécute les tâches de manière non bloquante, permettant à Node.js de gérer des milliers de connexions simultanément.

2 Modules et Gestion des Dépendances

1. **Expliquez le concept de module en Node.js. Quelles sont les principales différences entre les modules CommonJS et les modules ES6 ?**
   En Node.js, un module est un fichier JavaScript indépendant qui peut exporter des fonctions, objets, ou variables pour être utilisés dans d'autres fichiers. CommonJS utilise `require()` et `module.exports`, tandis qu'ES6 utilise les mots-clés `import` et `export`.

2. **Comment peut-on créer un module en Node.js ? Décrivez les étapes pour exporter et importer un module simple.**
   Pour créer un module, on peut définir des fonctions ou objets dans un fichier, puis utiliser `module.exports` pour les rendre disponibles. Par exemple :

   ```
   // math.js
   function add(a, b) {
       return a + b;
   }
   module.exports = add;
   ```

   Pour l'importer :

   ```
   const add = require('./math');
   console.log(add(2, 3)); // Affiche 5
   ```

3. **Qu'est-ce que NPM ? Décrivez son rôle dans un projet Node.js et expliquez la différence entre une dépendance locale et globale.**
   NPM (Node Package Manager) est un gestionnaire de paquets qui permet d'installer et de gérer les dépendances de Node.js. Une dépendance locale est installée dans le dossier du projet (`node_modules`), tandis qu'une dépendance globale est accessible à tous les projets et installée dans un emplacement global.

3 Gestion des APIs et Routage

1. **Expliquez ce qu'est une API REST. Pourquoi est-elle fréquemment utilisée avec Node.js ?**
   Une API REST est une interface permettant la communication entre des systèmes via des requêtes HTTP, en suivant des principes d'architecture REST. Elle est fréquemment utilisée avec Node.js pour construire des services web interactifs grâce à sa gestion efficace des requêtes HTTP.

2. **Quels sont les principaux types de requêtes HTTP utilisés dans les APIs ? Donnez un exemple d'utilisation de chacune.**
   Les principales requêtes HTTP sont :

   - **GET** : Récupérer des données (ex. obtenir les informations d'un utilisateur).
   - **POST** : Envoyer des données (ex. créer un nouvel utilisateur).
   - **PUT** : Mettre à jour des données (ex. modifier les informations d'un utilisateur).
   - **DELETE** : Supprimer des données (ex. supprimer un utilisateur).

3. **Qu'est-ce qu'un middleware dans Express.js ? Donnez un exemple de middleware pour l'authentification d'un utilisateur.**
   Un middleware est une fonction qui s'exécute entre la réception de la requête et l'envoi de la réponse. Un exemple pour l'authentification :

```
function authMiddleware(req, res, next) {
    if (req.isAuthenticated()) {
        return next();
    } else {
        res.redirect('/login');
    }
}
```

4 Asynchronisme et Gestion des Promesses

1. **Expliquez la différence entre une fonction synchrone et une fonction asynchrone. Donnez un exemple en Node.js.**
   Une fonction synchrone bloque l'exécution jusqu'à ce qu'elle soit terminée, tandis qu'une fonction asynchrone permet de continuer l'exécution sans attendre. Exemple :

```
// Synchrone
const result = fs.readFileSync('file.txt', 'utf8');

// Asynchrone
fs.readFile('file.txt', 'utf8', (err, data) => {
    if (err) throw err;
    console.log(data);
});
```

2. **Qu'est-ce qu'une Promesse en JavaScript ? Expliquez les états possibles d'une Promesse et leur signification.**

Une Promesse représente une opération asynchrone qui peut être dans un des états suivants :

- **Pending** : La promesse est en cours.
- **Fulfilled** : La promesse a réussi.
- **Rejected** : La promesse a échoué.

3. **Décrivez `async` et `await`. En quoi améliorent-ils la gestion des Promesses ? Donnez un exemple simple pour illustrer leur usage.**
`async` et `await` simplifient le code asynchrone en permettant de gérer des promesses comme du code synchrone. Exemple :

```
async function fetchData() {
    try {
        const data = await fetch('https://api.example.com/data');
        console.log(data);
    } catch (error) {
        console.error(error);
    }
}
```