Rapport TP1: Charité et Blockchain

DZIRI Samy

Étape 1 : Mise en place du serveur
Dans cette étape, j'ai cloné un fichier depuis GitHub, puis j'ai géré les deux principaux endpoints. J'ai également ajouté des logs pour suivre les requêtes et faciliter le débogage. Enfin, j'ai validé le serveur en utilisant Postman afin d'envoyer des requêtes et de vérifier les réponses."

Étape 2 : Lecture et gestion des fichiers JSON

Dans cette étape, j'ai implémenté la fonction `findBlocks` pour lire et retourner les données de la blockchain depuis un fichier JSON. J'ai créé un dossier `data` à la racine du projet, ainsi qu'un fichier `blockchain.json` contenant des données de test. J'ai ensuite écrit la méthode `findBlocks`, en utilisant la méthode `readFile` du module `fs/promises` pour lire le fichier. Les données sont converties au format JSON avant d'être retournées au client. Pour valider le bon fonctionnement, j'ai effectué des tests avec Postman et curl afin de m'assurer que le contenu du fichier est bien renvoyé en JSON.

Les principales difficultés rencontrées lors de l'implémentation ont concerné la gestion des chemins d'accès au fichier JSON. La première difficulté a été de définir correctement le chemin relatif du fichier en utilisant la constante `path`. Ensuite, un problème est survenu lorsque le fichier JSON n'a pas été trouvé, ce qui semblait être lié à une erreur dans le chemin d'accès. Ce problème a été résolu après avoir vérifié attentivement la structure des dossiers et corrigé le chemin relatif, ce qui a permis d'accéder au fichier sans problème.

Étape 3 :Création de nouveaux blocs

Dans cette étape, j'ai développé la fonction `createBlock()` pour ajouter de nouveaux blocs à la blockchain et les enregistrer dans le fichier `blockchain.json`. J'ai utilisé la fonction `uuidv4()` pour générer un identifiant unique pour chaque bloc, puis créé un bloc avec les champs id, nom, don, date et hash. Ensuite, j'ai lu le fichier `blockchain.json` et, en cas de fichier vide ou introuvable, j'ai réinitialisé la blockchain à un tableau vide. J'ai ajouté le nouveau bloc à la blockchain et mis à jour

le fichier JSON. La principale difficulté rencontrée a été la gestion des erreurs lors de la lecture du fichier JSON, car il pouvait être vide ou inexistant, ce qui causait des erreurs lors de l'ajout du bloc. Pour résoudre ce problème, j'ai ajouté des vérifications pour m'assurer que la blockchain était un tableau avant d'ajouter un bloc, et j'ai utilisé des logs pour suivre l'exécution et identifier les erreurs.

Étape 4 :Vers l'infini et au-delà !

Dans cette étape, j'ai ajouté un champ `hash` à chaque bloc de la blockchain, calculé à partir des données du bloc précédent à l'aide de l'algorithme de hachage SHA-256, à l'exception du premier bloc qui n'a pas de bloc précédent. J'ai réalisé la fonction `findLastBlock()` pour récupérer le dernier bloc de la blockchain et, si nécessaire, retourner `null` lorsque la blockchain est vide. Pour chaque bloc, j'ai utilisé le module `crypto` de Node.js pour calculer le hachage SHA-256, en convertissant les données du bloc en chaîne JSON. Lors de la création d'un bloc via une requête POST, j'ai récupéré les données envoyées par l'utilisateur, généré un identifiant unique avec `uuidv4()` et renseigné le champ `date`. Le bloc a ensuite été ajouté à la blockchain, et la nouvelle version a été sauvegardée dans le fichier `blockchain.json`. Après la création du bloc, j'ai renvoyé ce dernier au client pour confirmation. Les principales difficultés rencontrées étaient la gestion du calcul du hachage, la gestion du premier bloc sans hachage et la réécriture correcte de la blockchain dans le fichier JSON. Toutefois, une fois ces défis surmontés, le système fonctionne maintenant correctement, garantissant l'intégrité de la blockchain grâce au champ `hash`.