2024 - TP1

RAPPORT DÉVELOPPEMENT AVANCÉ

Mathéo BIENVENU

SOMMAIRE

1 - EXPLICATIONS DE MES CHOIX ET DIFFICULTÉS RENCONTRÉES

Etape 1 :

Au sein de cette étape, je n'ai pas été confronté à de sérieux problèmes. Il était nécessaire d'ajouter des "**console.log**" pour afficher un message dans la console, afin de vérifier que le serveur recevait correctement la requête et exécutait la condition adéquate dans notre structure "**switch**".

Etape 2 :

Lors de cette étape, j'ai entrepris le développement de la fonction findBlocks, en m'assurant de bien comprendre les attentes liées à cette fonction. Préalablement, j'ai créé un répertoire data et un fichier nommé "**blockchain.json**", dans lequel j'ai inséré le message fourni dans l'énoncé pour effectuer mes tests.

J'ai initialisé "**const path**", mais j'ai rencontré un souci : le chemin d'accès dans mon projet semblait incorrect. Depuis le fichier "**blockchainStorage.js**", pour accéder à "**blockchain.json**", je devrais normalement sortir du dossier src contenant blockchainStorage.js, puis me diriger vers le fichier souhaité, donc le chemin devrait être "**../data/blockchain.json**". Cependant, en utilisant la méthode readFile, le chemin partait de la racine du projet, nécessitant "**data/blockchain.json**". Bien que cela ne soit pas normal, j'ai décidé de continuer sans m'attarder sur ce problème, pensant qu'une réorganisation du projet pourrait le résoudre. Pour la fonction "**findBlocks()**", je n'ai pas rencontré de difficulté majeure : il suffisait d'utiliser "**readFile()**" en consultant la documentation.

Comme cette méthode est asynchrone, j'ai employé "**.then**" pour renvoyer la réponse et "**.catch**" pour gérer les erreurs avec "**NotFoundError()**". Le problème survenu ici était de ne pas réussir à renvoyer le résultat depuis ma fonction anonyme dans le "**.then**" de "**readFile()**", ce qui empêchait la propagation du résultat au serveur, rendant les requêtes **GET** infructueuses.

Etape 3 :

Pour élaborer la méthode “**createBlock()**”, j'ai d'abord examiné les instructions, puis j'ai simplement utilisé un “**console.log**” pour observer le format du contenu de la requête **POST** reçue. Cela m'a permis de décomposer facilement le contenu de la requête en plusieurs variables. Ensuite, j'ai importé la bibliothèque “**uuid**” et utilisé “**uuidv4**" pour générer un identifiant. Rapidement, j'ai complété ma fonction “**getDate()**” en y insérant simplement “**return Date()**”, me fournissant ainsi une date complète. J'ai ensuite créé un objet block contenant le contenu de la requête, la date et l'identifiant générés directement dans l'objet block.

Etape 4 :

Pour cette phase, j'ai débuté par le développement de “**findLastBlock()**”. Cette fonction commence par appeler “**findBlocks()**” pour récupérer tous les blocs. Cependant, les blocs sont retournés sous forme de chaîne de caractères, donc j'utilise “**JSON.parse()**” pour convertir cette chaîne en un tableau de blocs, me permettant ainsi de récupérer facilement le dernier bloc avec : “**const lastBlock = currentBlocksArray[currentBlocksArray.length - 1];**”

Avant de renvoyer le dernier bloc, je vérifie son existence, ce qui informe la fonction “**createBlock()**” si elle doit initialiser le hash pour le premier bloc. Je vérifie donc si le tableau de blocs est vide (taille égale à 0) : dans ce cas, je renvoie “**null**”, sinon je renvoie “**lastBlock**”.

De retour dans “**createBlock()**”, je commence par récupérer “**lastBlock()**” en appelant “**findLastBlock()**”, en veillant à utiliser await puisque c'est une fonction asynchrone.
Avec une condition simple, je vérifie si “**lastBlock**” est **null**. Si c'est le cas, je crée un bloc avec les données de la requête comme précédemment, mais maintenant nous avons un attribut hash en plus donc si “**lastBlock**” est **null**, cela signifie qu'aucun bloc n'a été créé auparavant, et je dois donc générer le premier hash. J'utilise pour cela ma variable “**monSecret**”, dont le contenu deviendra mon premier hash grâce à l'algorithme **sha256**.

Si **lastBlock** n'est pas **null**, je récupère simplement le bloc précédent, déjà obtenu avec “**findLastBlock()**”. J'utilise “**JSON.stringify()**” pour convertir ce dernier bloc en chaîne de caractères, que j'utiliserai ensuite pour générer le hash du prochain bloc, toujours avec l'algorithme **sha256**.
Une fois le nouveau bloc créé, je récupère l'intégralité de ma blockchain avec “**findBlocks()**”. Comme mentionné précédemment, cela me retourne ma blockchain sous forme de chaîne de caractères, que je transforme en tableau à l'aide de “**JSON.parse()**”. J'ajoute ensuite le nouveau bloc à la fin de mon tableau en utilisant **l'opérateur spread**, puis j'utilise “**writeFile()**” pour réécrire mon fichier avec le nouveau tableau de blocs. Enfin, je renvoie le bloc créé en réponse à la requête **POST**.

2 - AMÉLIORATIONS POSSIBLES

Je pense qu'il serait bénéfique de modifier “**findBlocks()**” pour qu'elle retourne directement la blockchain sous forme d'objet, ce qui éliminerait le besoin de “**JSON.stringify()**” dans server.js lors de l'envoi de la réponse, et éviterait également l'usage de “**JSON.parse()**” dans “**findLastBlock()**” et “**createBlock()**”.

Pour améliorer le système, j'aurais pu implémenter la fonction “**verifBlocks()**”, mais par manque de temps récemment, je ne l'ai pas réalisée.

Si je devais la concevoir, je commencerais par récupérer le premier bloc, en générerais un hash et vérifierais si le hash du bloc suivant (le deuxième, donc) correspond au hash obtenu. Il faudrait répéter cette vérification pour chaque bloc : si un hash ne correspond pas, la fonction renverrait “**false**”, si tous les hashes correspondent, cela signifierait qu'il n'y a pas de problème dans notre blockchain, et la fonction renverrait “**true**”.

Pour la fonction “**findBlock()**”, je parcourrais simplement mon tableau pour trouver le bloc correspondant à l'identifiant recherché. Ensuite, je vérifierais sa fiabilité en créant un hash avec le bloc précédent : si le hash obtenu est identique, le bloc est fiable; sinon, il ne l'est pas, et je renverrais un JSON indiquant que le bloc n'est pas fiable. Si le bloc est fiable, je le renverrais; s'il ne l'est pas, je renverrais le bloc avec un message indiquant son manque de fiabilité.

3 - CONCLUSION

En conclusion, j'ai trouvé ce TP relativement simple et n'ai pas rencontré de difficultés majeures concernant les tâches requises. Mes principaux défis étaient liés à des problèmes structurels du projet, comme la question du chemin d'accès au fichier "**blockchain.json**". Malgré cela, j'ai dû réfléchir à la manière d'aborder les différentes étapes, mais j'ai pu résoudre les problèmes rencontrés de façon assez fluide et rapide.