R6.A.05 Développement Avancée

TP1 : Charité et Blockchain - 27/01/2024

Etape 1 :

J'ai commencé par créer un dépôt local sur Github Desktop que j'ai nommé DevAvancee-TP1. Ensuite, j'ai ouvert ce dépôt sur PHPStorm et j'ai cloné le TP1 fourni par l'enseignant grâce à la commande

git clone https://github.com/laurentgiustignano/hachage-tp1.

Enfin, j'ai mis mon dépôt en ligne via Github Desktop afin de le voir apparaître sur GitHub.

Afin de tester le bon fonctionnement du projet, j'ai inséré deux console.log aux endroits appropriés (respectivement `console.log("get fonctionne")` | `console.log("post fonctionne")`). Ensuite sur Postman , j'ai lancé la requête http://localhost:3000/blockchain en GET (qui a fonctionné car le message a bien été affiché en console), puis en POST (qui n'a pas fonctionné car le message n'a pas été affiché en console).

Etape 2 :

J'ai étudié la structure du type Block préalablement défini par l'enseignant. Ensuite, j'ai codé la fonction findBlocks, dont le rôle est de récupérer l'ensemble de la blockchain et de la retourner au client sous forme de Json. Pour commencer, j'ai créé le fichier .json qui contiendra la blockchain en créant un dossier data à la racine du projet. Dedans, j'ai créé un fichier nommé blockchain.json en veillant à modifier la ligne 8 du fichier blockchainStorage.js pour initialiser le const path convenablement. Dans le fichier blockchain.json, j'ai placé un contenu Json pour tester la fonctionnalité ({"message" : "Bonjour à tous"}). À l'aide de la documentation du module node :fs/promises, j'ai effectué la lecture du fichier blockchain.json afin de retourner les valeurs lues au format Json.

Etape 3 :

J'ai développé la fonction createBlock() qui ajoute les blocs dans le fichier. La méthodologie la plus simple à appliquer est de créer un Block avec les nouvelles informations, reconstituer un tableau de Block avec tous les blocs existants et en rajoutant à la fin le nouveau Block. Enfin, la liste mise à jour est enregistrée dans un fichier via writeFile(path, JSON.stringify(newBlocks)), et la nouvelle liste est retournée. Cet ensemble sera enregistré dans le fichier. Dans un premier temps, la fonctionnalité de hachage n'a pas été implémentée. Le champ id a été généré à l'aide de la fonction uuidv4() du module uuid. J'ai codé la méthode getDate() dans le fichier src/divers.js. Elle m'a permis d'obtenir le timestamp au format demandé. Le champ nom et don a été complété avec les valeurs transmises lors de la requête POST (avec mon prénom par exemple sur Postman).

Etape 4:

La dernière partie à développer consiste à ajouter le champ hash à chaque bloc. Hormis le premier enregistrement, tous les nouveaux blocs doivent récupérer le bloc précédent, en calculer la valeur de hachage de l'équivalent en string avec l'algorithme sha256, et insérer cette valeur comme champ hash. J'ai donc écrit la fonction findLastBlock() qui retourne un objet Block ou null. À l'aide de la classe Hash du module node :crypto, j'ai affiché la valeur d'un sha256 pour une chaine de caractère et vérifier à l'aide du site Internet (https://emn178.github.io/online-tools/sha256.html) l'exactitude de la valeur trouvé.