ÉNONCE TRAVAUX PRATIQUE – SUJET 1

Charité et Blockchain

Thèmes

- Les promises
- Le module **node :fs**
- Le module **node :crypto**

Présentation

Le principe de la Blockchain consiste à conserver et transmettre des informations de manière transparente et en sécurité. En s'appuyant sur des fonctions de hachage, comme par exemple avec l'algorithme **sha256**, les données peuvent être certifiées. Elles constituent un bloc, et chaque bloc est certifié en se basant sur le bloc précédemment certifié.
Nous allons utiliser ce principe pour enregistrer, dans le cadre d'une levée de fond, les différentes sommes transmises par les participants. Ces enregistrements s'effectueront dans un fichier **Json** qui contiendra le nom du donateur, la somme versée, la date de la transaction, et un identifiant. A ses informations s'ajoutera la valeur de hachage du bloc précédent. Cet ensemble formera le nouveau bloc qui s'enregistrera en fin de fichier.

```
{
  "id": "0b05ea1c-a17f-411a-9bb6-c31130b3e212",
  "nom": "Alan Turing",
  "don": 1234,
  "date": "20230802-16:31:30",
  "hash": "b28c94b2195c8ed259f0b415aaee3f39b0b2920a4537611499fa044956917a21"
},
```

Figure 1- Exemple d'enregistrement d'un bloc

Le travail à réaliser durant la séance ne portera uniquement sur l'ajout d'un nouveau bloc et la récupération de la chaine. Le travail de vérification de l'intégrité de la chaine ou d'un bloc est subsidiaire. De même, l'API est déjà fonctionnelle. Il n'est pas nécessaire d'en comprendre les détails pour la réalisation du TP.
Remarque : Si certains souvenir du JavaScript sont un peu lointain, vous trouverez ici (https://quickref.me/javascript) un des nombreux cheatsheet JavaScript disponible sur Internet.

Étape 1 – état des lieux

Un dépôt du projet est à récupérer sur Github. Vous pouvez récupérer le squelette du code depuis **PhpStorm** en choisissant : **Get from Version Control**. Ou bien dans votre répertoire de travail, avec la commande **git clone https://github.com/laurentgiustignano/hachage-tp1.**
Une fois le projet ouvert, vous observerez dans le fichier **src/server.js** la présence d'une fonction **createServer()**. Dans le corps de cette fonction, la structure **try/catch** contient le code qui effectue la différentiation de méthode sur le **endpoint** de l'application http://localhost:3000/blockchain.

- Avant le mot clé **break**, ajoutez une instruction permettant de vérifier le bon fonctionnement du serveur http en affichage en console un message identifiable. Vous pouvez l'outil **Postman** (ou équivalent) pour valider les deux méthodes (**GET/POST**).

Étape 2 – apprenons à lire

Dans le fichier **src/blockchainStorage.js**, une description commençant à la ligne 10, permet de définir un type **Block** en détaillant sa composition. Ainsi, **PhpStorm** pourra afficher la conformité de vos retours de fonction et les **JSDoc** que j'ai déjà spécifiés. Vous devrez évidemment les respecter.

La première fonction à développer sera **findBlocks()**. Son rôle est de récupérer l'ensemble de la blockchain et de la retourner au client sous forme de **Json**. Pour commencer, il faut créer le fichier qui contiendra la blockchain.

- Créez un dossier **data** à la racine du projet. Dedans, créez un fichier nommé **blockchain.json**
- Modifiez la ligne 8 pour initialiser **const path** convenablement
- Dans le fichier, placez un contenu **Json** à votre convenance qui permettra de tester la fonctionnalité. Par exemple : **{"message" : "Bonjour à tous"}**

Remarque : Notez qu'un enregistrement Json nécessite des guillemets pour les clés, alors qu'un objet en JavaScript n'en requiert pas.

- À l'aide de la documentation du module **node :fs/promises**, effectuez la lecture du fichier **blockchain.json**, puis retournez les valeurs lues au format **Json**. A présent, votre serveur doit renvoyer le contenu **Json** lors d'un appel **GET /blockchain**.

Étape 3 – une brique après l'autre

Vous allez développer 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**. Et cet ensemble sera enregistré dans le fichier. Dans un premier temps, la fonctionnalité de hachage ne sera pas implémentée.

- Le champ **id** sera généré à l'aide de la fonction **uuidv4()** du module **uuid**. Vous pourrez importer le module dans votre code après avoir installé le module avec **npm**.
- La méthode **getDate()** est à compléter dans le fichier **src/divers.js**. Elle vous permettra d'obtenir le timestamp au format demandé.
- Le champ **nom** et **don** seront complété avec les valeurs transmises lors de la requête **POST**. Dans **Postman**, vous pourrez spécifier dans le corps du message les données au format **Json** contenant ces deux clés.

Étape 4 – « Vers l'infini et au-delà ! »

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**.

- Écrire la fonction **findLastBlock()** qui retourne un objet **Block** ou **null**.
- À l'aide de la classe **Hash** du module **node :crypto**, afficher la valeur d'un **sha256** pour une chaine de caractère. Vous pouvez vérifier à l'aide du site Internet (https://emn178.github.io/online-tools/sha256.html) l'exactitude de la valeur trouvée, si vous n'avez pas l'outil **shasum** sur votre ordinateur. Si cette fonctionnalité est validée, vous pouvez encoder le bloc retourné par **findLastBlock()** et l'insérer dans le nouveau bloc.
- Faites le nécessaire pour que le nouvel enregistrement soit retourné au client.

Pour aller plus loin…

Vous pouvez développer de nouvelles fonctionnalités pour vérifier l'intégrité de la chaine.

- **verifBlocks()** : qui s'assure de l'intégrité de la chaine. En modifiant une valeur dans la blockchain, le retour de la fonction doit être **false**. Si les blocs restent liés par leurs valeurs de hachage, le retour sera **true**.
- **findBlock(id)** : récupère un bloc particulier à partir de son id. Si le bloc n'est plus fiable, un retour **Json** affichant cet état doit être retourné

Bon courage…