TP Blockchain

1. Objectifs

L'objectif de ce TP va être de **créer un système de blockchain**.
Chaque participant a la blockchain va découvrir quels sont les acteurs de la blockchain grâce au réseau **UDP**, et ils communiqueront ensemble via **HTTP** par la suite.

Nous n'allons pas créer de crypto-monnaie ni autre, le but est simplement de voir comment fonctionne un réseau **Peer-to-Peer**, nécessitant de se mettre d'accord sur les informations communes.

2. Architecture

Pour cela, nous utiliserons une représentation très basique de la blockchain :

- Tout les participants du réseau sont des mineurs
- Notre **Proof-Of-Work** consiste à vérifier que le **hash** d'un block ajouté a la chaine soit bien **divisible par 7**
- Lorsqu'un client du réseau reçoit un nouveau nœud, il décide de l'accepter s'il le considère comme bon, ou de le refuser sinon
    - Un nœud est bon si la PoW est validée, et si il n'y a pas de nœuds avec un hash plus petit en attente d'être ajoutés
- Toutes les requêtes (acceptation, refus, validation de block) seront envoyés via HTTP.

2.1 La blockchain

La blockchain se situera dans le fichier ***models/Blockchain.js***.
Ce sera une structure qui sera copiée chez tous les participants du réseau.
Elle sera composée d'une liste de **Block**.
Il ne peut exister qu'une seule instance de la blockchain par participant, et le contenu doit être identique pour chaque participant du réseau.

2.2 Les blocks

Un block sera le composant principal de la blockchain.
Un block contient plusieurs champs :

- Un **index**, qui s'incrémente à chaque block
- Un **timestamp**, qui indique le moment de création du block

- Un champ **data**, qui contient :
  - L’identifiant de l’envoyeur
  - L’identifiant du receveur
  - Le message transmis
- Le **hash du bloc précédent**
- Son **propre hash**
- Un champ **nonce**, qui indique le nombre de tentatives pour arriver au bon hash.

Nous utiliserons le module ***crypto-JS*** pour hacher facilement notre nœud. Notre besoin se limitera a la fonction *SHA256(obj)* qui permet d’obtenir le hash d’un objet via le protocole SHA256.

3. Communication

3.1 Découverte des participants via UDP

Lorsqu’un client rejoint le réseau, elle émet une requête **broadcast** en **UDP** pour découvrir les autres participants (**PING**).
Chaque participant du réseau devra lui répondre par un **PONG**, ce qui permettra au client de construire une table de routage.

Si le client ne reçoit aucun **PONG** dans un cours laps de temps, dans ce cas il se considérera comme le premier nœud du réseau, et devra créer le **block genesis**.

S’il reçoit un **PONG**, il va demander la blockchain à un des participants via une requête **HTTP**.

3.2 Communication via HTTP

Pour communiquer entre eux, chaque client devra exposer une API avec plusieurs routes.
Les routes sont les suivantes :

- ***POST /block/check*** : permet de recevoir un block de la part d’un autre client. Le receveur effectuera une analyse du bloc. Si le bloc n’est pas bon, il enverra directement un refuse.
  Si le block est correct, il l’ajoutera à sa liste de block en attente de validation (nécessité d’avoir les réponses de chaque client) et enverra un Accept à chaque participant du réseau pour ce block.
  La réponse sera dans tout les cas un code 200.
- ***POST /block/accept*** : permet de recevoir un Accept de la part d’un client pour un block donné dans le body ({block : {…}). Si le block est accepté par toute la table

de routage du receveur, il peut l'ajouter à sa propre chaine.
La réponse sera dans tout les cas un code 200.

- ***POST /block/refuse*** : permet de recevoir un Refuse de la part d'un client pour un block donné dans le body ({block : {…}). Dès qu'un refuse est reçu, il faut supprimer le block en question de la liste des blocks en attente.
  La réponse sera dans tout les cas un code 200.
- ***GET /blockchain*** : permet d'obtenir la blockchain au format JSON.

Nous utiliserons le fichier ***httpServer.js*** pour remplir ces fonctions.
Pour envoyer des requêtes HTTP, nous utiliserons ***node-fetch***.

4. Le minage

Chaque participant au réseau pourra décider de miner un block (c'est à dire d'envoyer un message de manière aléatoire a n'importe qui du réseau, sous forme de block avec un hash correct).
La liste des messages possibles est dans constants.js, en prendre un aléatoirement fait largement le boulot (c'est simplement histoire d'avoir une data à envoyer).

Le minage étant une opération intensive en calcul, il ne faudra pas blocker le main thread de node.js.
Par conséquent, nous l'exécuterons dans un thread a part, grâce au module worker_threads de Node.js.

Le minage se déclenchera de manière aléatoire dans un intervalle de secondes (entre 5 et 10 secondes sans miner, par exemple).
Un client peut miner uniquement s'il n'est pas déjà en train de miner, et s'il n'est pas en train d'attendre la validation de son block miné par le reste du réseau.

Une fois terminé, on vérifie que le block miné est correct, et qu'il est mieux que les block actuellement en attente.
Si c'est le cas, on envoie un refuse pour les blocks en attente, puis on demande à chaque participant de checker le nouveau block.
Si ce n'est pas le cas, alors le block est détruit, et le client recommencera a miner plus tard.
S'il y a consensus sur l'acceptation du block, alors il sera ajouté a la blockchain.

5. Ressources utiles

Documentation du module UDP : https://nodejs.org/api/dgram.html

Documentation du module crypto-js : https://cryptojs.gitbook.io/docs/

Documentation de express : https://expressjs.com/fr/4x/api.html

Documentation des worker_threads :https://nodejs.org/api/worker_threads.html

Documentation de node-fetch pour envoyer des requêtes HTTP : https://www.npmjs.com/package/node-fetch

Pour lancer plusieurs nœuds sous docker, il faut d'abord construire l'image docker

```
docker build -t blockchain-node .
```

Une fois construite, on peut lancer docker compose

```
docker compose up
```

Vous êtes libre de tester avec autant de nœuds que vous voulez (je vous conseille d'en mettre seulement 2 au départ, et 4-5 si vous souhaitez voir un peu plus de mouvement).