TP - Implémentation d’une blockchain

Introduction

L’objectif de ce lab est de vous faire mettre en pratique les concepts théoriques que vous avez découverts durant le cours. Pour ce faire, vous allez devoir implémenter une blockchain théorique: **IsiChain**.

Rappel

La blockchain a différents algorithmes de consensus.

Dans les blockchains utilisant le Proof-of-Work (PoW), les mineurs doivent investir une grande quantité de puissance de calcul afin de trouver une chaîne de hachage qui corresponde à un ensemble donné de contraintes.

Dans Bitcoin, les mineurs sont tenus de trouver un hachage tel qu’il existe un certain nombre de zéros non significatifs au début du hachage.

- Un mineur trouve en permanence de nouvelles valeurs de hachage d'un bloc en modifiant un nombre, qui est haché avec le bloc, appelé nonce.
- Le nombre de zéros dans la valeur de hachage est appelé difficulté. C’est la difficulté qui est attribuée à des retards aussi longs dans la génération de blocs.

```
let block = {"random":"data"};
let nonce = 1;
let difficulty = 5;
let blockHash = hash(block, nonce, difficulty);
// blockHash = 6B86B273FF34FCE19D6B804EFF5A3F5747ADA4EAA22F1D49
C01E52DDB7875B4B
```

Puisque blockHash ne respecte pas les contraintes, le mineur incrémente le nonce et réessaie. Le mineur doit le faire jusqu’à ce qu’il reçoive quelque chose comme

ceci : 00000273FF34FCE19D6B8

Implémentation du prototype

Voici la liste des étapes sur lesquelles vous allez travailler pour construire votre prototype de **IsiChain**:

- Étape 0: Initialiser le projet
- Étape 1: Création d'un Bloc
- Étape 2: fonction de hachage
- Étape 3: Mining (sans PoW) + tests
- Étape 4: Création de la blockchain + tests
- Étape 5: Validation d'une chaîne de blocs
- Étape 6: Web API (exposé la blockchain en web)
- Étape 7: P2P
- Étape 8: PoW consensus

Bien sûr il faudrait continuer la construction par exemple en ajoutant un système de Wallet, etc. mais pour ce qui nous intéresse, et le temps imparti, dans le cadre de ce cours, nous allons nous limiter à ces fonctionnalités.

Étape 0: Initialiser le projet

Travail à effectuer:

```
> npm init -y
> npm install nodemon --save-dev # pour monitorer les changements et redé
marrer le serveur
> git init
```

Étape 1 - création d'un Bloc

La Blockchain est composée de blocs, alors créons une classe de blocs. Le bloc a 4 propriétés principales:

- timestamp - le temps de création du bloc en millisecondes
- lastHash - hash du dernier bloc de la chaîne
- hash - hash du bloc actuel
- data - données dans le bloc ou les transactions

Travail à effectuer:

Créez un nouveau fichier `block.js` dans votre répertoire racine et créez une classe avec ces propriétés.

```
+ construction(timestamp, lastHash, hash, data) {...}
+ toString(){...}
+ static genesis(){...}
```

Étape 2 - fonction de hachage

Pour générer une valeur de hachage, nous avons besoin d'un paquet de nœuds appelé `crypto-js`, installons-le.

```
> npm i crypto-js --save
```

Importez la fonction `sha-256` à partir de ce module.

```
> const SHA256 = require('crypto-js/sha256');
```

Étape 3 - Dummy Mining + tests

Il nous faut maintenant une fonction nous permettant de créer un bloc... dans le cadre du PoW, on "mine" un bloc. Pour le moment, nous allons le faire sans fournir de preuve de travail.

Travail à effectuer:

- Écrire une fonction `mineBlock(lastBlock, data)`
- Installer une librairie de test:

```
> npm i jest --save-dev
```

Ajout un fichier **block.test.js** contenant le code suivant:

```
const Block = require('./block')

describe("Block", () => {
  let data, lastBlock, block

  beforeEach(() => {
    data = 'bar'
    lastBlock = Block.genesis()
    block = Block.mineBlock(lastBlock, data)
  })

  it("sets data to block", () => {
    expect(block.data).toEqual(data)
  })

  it("sets the lastHash to the value of the last block hash", () => {
    expect(block.lastHash).toEqual(lastBlock.hash)
  })
})
```

Ajouter au fichier **package.json**

```
"test": "jest --watchAll"
```

Vérifier que les tests fonctionnent en exécutant:

```
> npm run test
```

-

Étape 4 - Création de la Blockchain

La chaîne de bloc contient des blocs et s'initialise avec un premier bloc.

Travail à effectuer:

- Créer un fichier `blockchain.js` contenant:

```
+ Constructeur
+ addBlock(data){...}
```

- Ajouter un fichier `blockchain.test.js` contenant:

```
const Blockchain = require('../blockchain')
const Block = require('../block')

describe("Blockchain", () => {
 let blockchain

 beforeEach(() => {
  blockchain = new Blockchain()
 })

 it("starts with genesis block", () => {
  expect(blockchain.chain[0]).toEqual(Block.genesis())
 })

 it("add a new block", () => {
  const data = 'foo'
  blockchain.addBlock(data)

  expect(blockchain.chain[blockchain.chain.length - 1].data).toEqual(dat
a)
  })
})
```

- Effectuer les tests.

Étape 5: Validation d'une chaîne de blocs

Pour valider une chaîne, il faut vérifier qu'aucun bloc n'a été altéré. Pour ce faire, il est nécessaire de prendre chaque bloc de la chaîne et de recalculer son hash et de le comparer avec le hash contenu dans le bloc suivant.

Travail à effectuer:

- Implémenter les fonctions suivantes:

```
+ blockHash(block)
+ isValidChain(chain)
+ replaceChain(chain) // pour rappel: la chaîne valide la plus longue l'emp
orte
```

- Remplacer le contenu du fichier `blockchain.test.js` par:

```
const Blockchain = require('../blockchain')
const Block = require('../block')

describe("Blockchain", () => {
  let blockchain
  let blockchain2

  beforeEach(() => {
    blockchain = new Blockchain()
    blockchain2 = new Blockchain()
  })

  it("starts with genesis block", () => {
    expect(blockchain.chain[0]).toEqual(Block.genesis())
  })

  it("add a new block", () => {
    const data = 'foo'
    blockchain.addBlock(data)
```

```
    expect(blockchain.chain[blockchain.chain.length - 1].data).toEqual(data)
  })

  it("validates a valid chain", () => {
    blockchain2.addBlock('foo')
    expect(Blockchain.isValidChain(blockchain2.chain)).toBe(true)
  })

  it("invalidates a corrupted genesis block", () => {
    blockchain2.chain[0].data = 'corrupted'
    expect(Blockchain.isValidChain(blockchain2.chain)).toBe(false)
  })

  it("invalidates a corrupt chain", () => {
    blockchain2.addBlock('foo')
    blockchain2.chain[1].data = 'not foo'
    expect(Blockchain.isValidChain(blockchain2.chain)).toBe(false)
  })
})

it("replaces the chain with a valid chain", () => {
  blockchain2.addBlock('goo')
  blockchain.replaceChain(blockchain2.chain)

  expect(blockchain.chain).toEqual(blockchain2.chain)
})

it("does not replaces the chain with less than or equal chain", () => {
  blockchain.addBlock('foo')
  blockchain.replaceChain(blockchain2)

  expect(blockchain.chain).not.toEqual(blockchain2.chain)
})
```

- Effectuer les tests

-

Étape 6: Web API

Il nous faut maintenant exposer les services de IsiChain via une API WEB.

Travail à effectuer

- Exposer la blockchain avec deux points d'entrés (vanilla, express ou ce que vous voulez; on se référera dans la suite à une implémentation express):
  - `GET /blocks` ⇒ retourne la chaîne de l'utilisateur
  - `POST /mine` ⇒ envoie les données dans une requête pour construire un nouveau bloc
- Effectuer des tests:
  - Soit en créant un nouveau fichier de test
  - Soit en utilisant postman

Synthèse Étape 1 à 6

Nous avons:

- ☐ Créer une blockchain
- ☐ Ajouter des fonctionnalités de validation
- ☐ Rendu le tout disponible via une API représentant un nœud

Maintenant que le cœur de notre ledger est effectif, il nous faut passer à la prochaine étape: peer-to-peer.

Étape 7: Peer-to-Peer P2P

Nous pouvons maintenant interagir avec la blockchain, mais nous n'avons toujours qu'un seul utilisateur. Pour avoir plusieurs pairs, nous allons créer un serveur P2P et connecter ces pairs à l'aide de WebSockets. Chaque pair communique avec un autre pair et transmet des données.

Travail à effectuer - Connexion des nœuds

- Installer une librairie de websocket

```
npm i ws --save #websockets
```

- Créer un fichier `p2p-server.js` dans `app/`
- Créer une classe `P2PServer` contenant

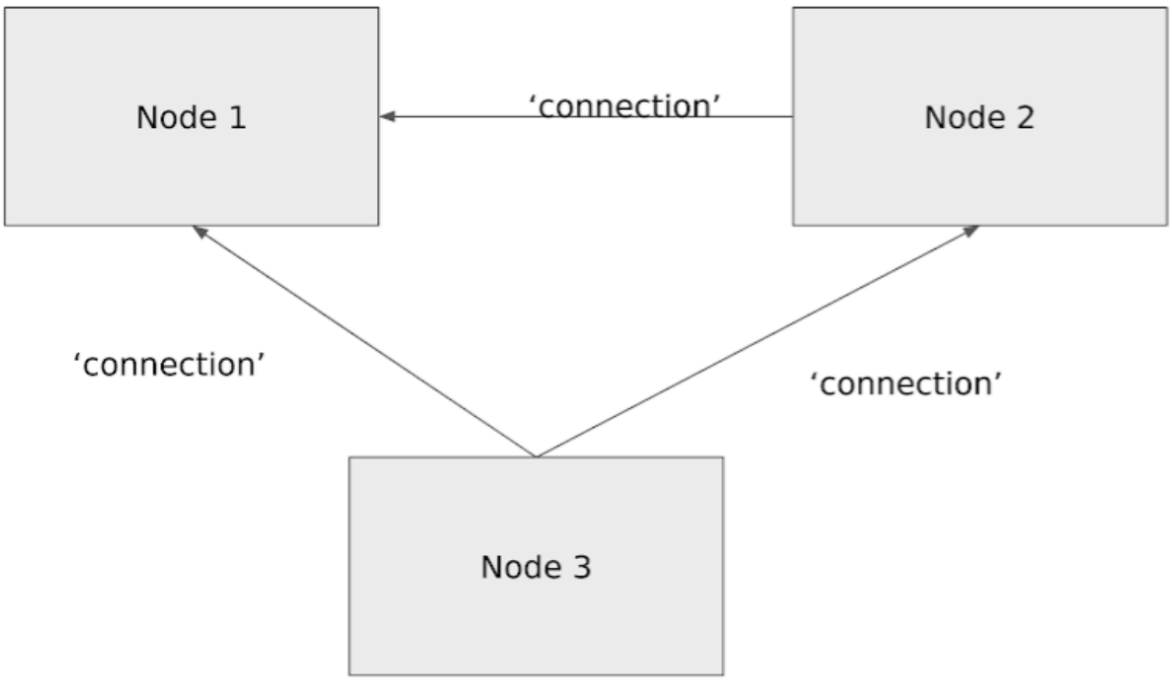
```
+ constructor(blockchain) // création de blockchain et sockets []
+ listen() // crée le serveur && on 'connection' handling
+ connectSocket(socket) // ajouter le socket au tableau des sockets et en voyer la blockchain
+ connectToPeers() // se connecter aux pairs définis dans les variables d'environnement
```

- Dans `app/index.js`, créer une instance du serveur `p2pserver`, y passer l'instance de la blockchain et démarrer le serveur `p2pserver`.

```
npm run dev
```

- Exemples de commandes pour exécuter les pairs:

```
HTTP_PORT=3002 P2P_PORT=5002 PEERS=ws://localhost:5001 npm run dev
HTTP_PORT=3003 P2P_PORT=5003 PEERS=ws://localhost:5002,ws://localhost:5001 npm run dev
```



Travail à effectuer - Échange de Ledger

Échange de ledger : Les nœuds devraient s'envoyer leurs versions de la blockchain, pour ce faire:

- Ajouter une fonction `messageHandler` qui prend une socket en parallèle, se rajoute sur l'événement 'message' et appelle la fonction de remplacement de chaîne dans `messageHandler` (`this.blockchain.replaceChain(data)`).
- Créer deux fonctions utilitaires:

  ☐ `sendChain(socket)` permet d'envoyer notre blockchain à un socket (= autre nœud)

  ☐ `syncChain()` sera utilisé pour envoyer à tous les sockets notre blockchain. Sera utilisé dans `index.js` après la création d'un nouveau nœud (/min)

Étape 8: PoW Algorithm

Pour rappel, la difficulté permet de définir le nombre de zéros de tête (à gauche) dans le hash. Un hachage avec les zéros de tête égaux à la difficulté est un hachage valide.

Par exemple, pour un bloc de la blockchain de difficulté = 6, le hachage du bloc doit avoir des zéros de tête égaux à la difficulté.

Pour générer de nouveaux hachages, nous devons modifier une valeur dans le bloc afin de donner un hachage correspondant différent d'avant. Pour cela, il existe une variable dans le bloc appelée nonce.

Le nonce est utilisé avec les données de bloc pour générer un nouveau hachage de bloc et est également stocké dans le bloc. Les mineurs doivent ajuster ou incrémenter le nonce et générer à nouveau du hachage jusqu'à ce que nous ayons un hachage avec un nombre de zéros non significatif.

C'est ce qui nécessite une énorme puissance de calcul. C'est ce qu'on appelle l'exploitation minière.

Travail à effectuer

- Ajouter une constante

```
const DIFFICULTY=4
```

- Ajouter le nonce dans les méthodes:
  - `constructor()`
  - `toString()`
  - `hash()`
  - `blockHash()`
- Créer la fonction `mineBlock()` ⇒ va créer une variable locale nonce

Comment implémenter la difficulté dynamique?

En diminuant la difficulté du bloc actuel, si le temps nécessaire à la génération de ce bloc a été supérieur à un certain temps, nous l'augmenterons de la même manière dans le cas contraire.

Pour vérifier si la génération de blocs précédente était plus rapide ou plus lente, nous allons créer une constante appelée `MINE_RATE`, qui correspond au temps

supplémentaire requis en millisecondes pour créer le bloc. Considérez ceci comme un seuil.

Travail à effectuer

- Pour garder notre code propre, déplaçons les deux constantes dans un fichier `config.js` et les demandons dans le fichier `block.js`.
  - Modifier: `constructor`, `toString`, `genesis`, `hash`, `hashBlock`
- Créer une fonction `adjustDifficulty(lastBlock, currentTime)`
- Modification de `mineBlock` pour utiliser `let { difficulty } = lastBlock;`
- Tester le bon fonctionnement en rajoutant les cas de tests suivants au fichier `blok.test`:

```
it('lower difficulty for a slower generated block', () => {
  expect(Block.adjustDifficulty(block, block.timestamp + 30000)).toEqual
(block.difficulty - 1)
})

it('raise difficulty for fast generated block', () => {
  expect(block.adjustDifficulty(block, block.timestamp + 1)).toEqual(block.
difficulty + 1)
})
```

Partie Optionnelle: Wallet

Pour arriver jusqu'aux Wallet, il va falloir accomplir certaines étapes:

- Implémenter les transactions (avec adresses et montant)
- Avoir des pools de transactions
- Signer les transactions (création des clefs, etc.)
- Mettre en place les APIs pour faire des transactions, créer une adresse, etc.
- Mettre en place une interface de consultation de la blockchain et de passage d'ordres