SUPINFO
International University

SupBank

DOCUMENTATION TECHNIQUE

NATHAN MERCIER
CHARLES EMIEUX
JULIAN ZERBIB
THEO ROUSSET

TABLE DES MATIERES

BIBLIOTHEQUES UTILISEES

GSON

Gson est une bibliothèque open-source permettant de convertir un objet Java en Json et inversement.

BOUNCY CASTLE

Il s'agit d'une librairie java complétant Java Cryptographic Extension (JCE). Elle permet le chiffrement, déchiffrement, signature et vérification des données dans le traitement cryptographique.

JAVA SWING

Java Swing est une librairie permettant le développement d'interface graphique pour les utilisateurs (GUI). Les composants Swing sont utilisés avec la « Java Foundation Classes » (JFC).

EXPLICATION DES CHOIX D'IMPLEMENTATION

MECANISME DE CONSENSUS

L'algorithme de consensus facilite la vérification et la validation des informations. Cela garantie l'authenticité des transactions enregistrées sur la blockchain.

Cet algorithme suit des règles établies dans le protocole et permet d'assurer le fonctionnement du système de manière décentralisé.

Nous avons choisi POW (Proof Of Work) car c'est le plus adapté pour gérer notre blockchain qui utilise la fonction de hachage SHA-256 et le plus sécurisé.

```java
public static String applySha256(String input){
    try {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hash = digest.digest(input.getBytes("UTF-8"));
        StringBuffer hexString = new StringBuffer(); // This will contain hash as hexidecimal
        for (int i = 0; i < hash.length; i++) {
            String hex = Integer.toHexString(0xff & hash[i]);
            if(hex.length() == 1) hexString.append('0');
            hexString.append(hex);
        }
        return hexString.toString();
    }
    catch(Exception e) {
        throw new RuntimeException(e);
    }
}
```

Fonction utilisée pour le mécanisme de consensus

VALIDATION DES BLOCS

Afin de valider les blocs avant l'enregistrement dans la blockchain, on doit vérifier qu'il suit les règles suivantes :

- L'index du bloc doit suivre l'index du bloc précédent
- Le previousHash doit correspondre au hash du bloc précédent
- Le hash du bloc est bien valide

On suit donc ces règles avant de valider chaque transaction et donc chaque bloc.

```
public static Boolean isChainValid() {
    Block currentBlock;
    Block previousBlock;
    String hashTarget = new String(new char[difficulty]).replace('\0', '0');
    HashMap<String,TransactionOutput> tempUTXOs = new HashMap<String,TransactionOutput>();
    tempUTXOs.put(genesisTransaction.outputs.get(0).id, genesisTransaction.outputs.get(0));

    for(int i=1; i < blockchain.size(); i++) {

        currentBlock = blockchain.get(i);
        previousBlock = blockchain.get(i-1);

        if(!currentBlock.hash.equals(currentBlock.calculateHash()) ){
            System.out.println("#Current Hashes not equal");
            return false;
        }

        if(!previousBlock.hash.equals(currentBlock.previousHash) ) {
            System.out.println("#Previous Hashes not equal");
            return false;
        }

        if(!currentBlock.hash.substring( 0, difficulty).equals(hashTarget)) {
            System.out.println("#This block hasn't been mined");
            return false;
        }
```

Fonction validant les blocs.

TOKEN EARNING

Pour le minage de monnaie, nous avons créé une méthode nommée « mineblock ». Cette dernière prend comme paramètre un entier appelé « difficulty » qui permet de changer la difficulté de minage d'un bloc (plus la difficulté est grande, plus le temps de minage sera long).

```
public void mineBlock(int difficulty) {
    merkleRoot = StringHash.getMerkleRoot(transactions);
    String target = StringHash.getDifficultyString(difficulty);
    while(!hash.substring( 0, difficulty).equals(target)) {
        nonce ++;
        hash = calculateHash();
    }
    System.out.println("Block Mined!!! : " + hash);
}
```

Fonction permettant le minage des blocs.