Spécification Interface fichier

Solène CHATEAU - Arthur DESOUCHES

Jérémy GERMANICUS - Théo SEASSAL

Département MIDO
Année universitaire 2017 – 2018
M2 IF

Version	Date	Modifications
V 1.0	*15/02/2018*	*Création du document*
V 2.0	*21/02/2018*	*Ajout des spécifications de tests*

I – Description

1- Contexte

À la fois monnaie cryptographique et véritable alternative aux monnaies traditionnelles, Le Bitcoin présente une innovation majeure en proposant une nouvelle alternative au modèle financier actuel.

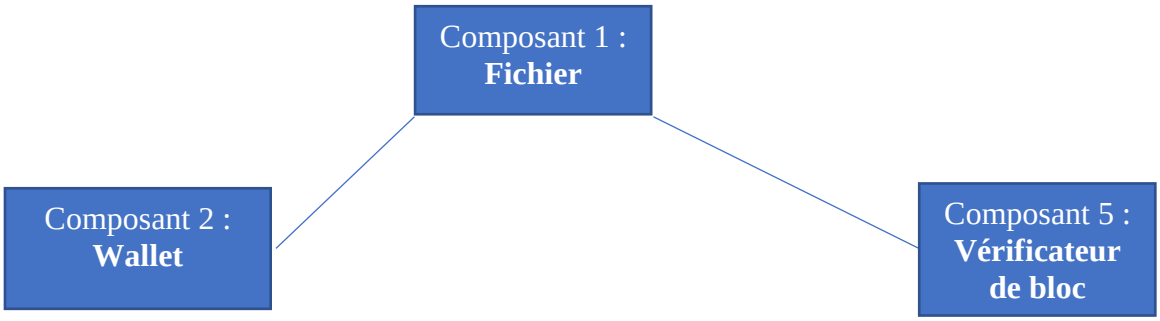
Le nom Bitcoin regroupe un système de paiement, une monnaie et une infrastructure de notariat électronique, capable de protéger l'intégrité des données qui y transitent.

Bitcoin est indépendant de tout organisme, banque, banque centrale ou état. Entièrement libre et transparent, il est une infrastructure communautaire et open-source. Le grand apport du Bitcoin est de nous offrir ce « livre magique » avec la technologie blockchain.

En appréhendant les concepts et mécanismes du Bitcoin, nous nous efforcerons de mettre en place un système de paiement peer-to-peer à l'image du système Bitcoin inventé par Satoshi Nakamoto dans ce cours de programmation par composants.

L'interface fichier est le premier composant constituant la blockchain. C'est lui qui définit sa représentation.

2- Schéma des interactions avec les composants connexes



3- Organisation

3.1- Fichier

Le fichier sera au format .txt. Il contiendra un ensemble de blocs et chaque bloc contiendra des transactions.

Une transaction sera un message avec une structure uniformisée. Une transaction enregistre une opération de transfert entre 2 utilisateurs. Chaque transaction est unique et non modifiable. Elle possède donc :

- Un identifiant
- Le numéro de bloc ou elle a été validée
- Une date
- L'identifiant de l'initiateur de la transaction
- L'identifiant du destinataire
- Le montant
-

Une structure de donnée pouvant être privilégié pour le format de la transaction peut être le JSON :

```
{
        "Id" : "1",
        "date" : "2018-02-15 12 :00 :00",
        "Bloc" : "1",
        "ID_init" : "2",
        "ID_recev" : "3",
        "montant" : "100"
}
```

3.2- Bloc

Les transactions sont enregistrées dans des blocs. Les blocs sont eux-mêmes chainés les uns aux autres pour former la blockchain. Un bloc est composé de deux parties : une en-tête et un corps. L'en-tête permet de restituer le jeton d'horodatage. L'entête est une empreinte numérique qui authentifie un bloc comme un maillon de la blockchain à l'aide de diverses informations. Le corps contient l'ensemble des transactions effectué sur ce bloc.

Un bloc possède un identifiant, un hash, le nombre de transactions, son code de hachage et celui du bloc précédent, une date de création, la version du hash, la somme des transactions et une liste des transactions.

```
{
	"Identifiant" : "24z",
	"NombreDeTransaction" : 45,
	"Hashes" : {
		"Hash" : "5abfd75c7b93e0e948a5380ee908f79",
		"BlocPrecedent" : "5sds85abfac7b93e08a5380ee948a53802"
	},
	"DateDeCreation" : "2018-02-15 12:00:00",
	"Version" : "0x2000",
	"Transactions" : [
		{transaction_1},
		{transaction_2}
		{transaction…}
	]
}
```

4- Fonctionnalités

4.1- Lecture du fichier

Cette fonction effectue une lecture du fichier, elle lit l'ensemble du fichier et identifie chaque bloc. La fonction prendra en paramètre le chemin d'accès au fichier et retournera la liste chainée des blocs.

Signature : std :: vector<Bloc> FileInterface (std :: string file)

4.2- Insertion d'un bloc après un bloc donné

Cette fonction a pour objectif d'insérer un bloc dans la blockchain et donc dans le fichier. Elle prend donc en paramètre le bloc à insérer et l'identifiant, ou le hash, du bloc précédent le bloc à insérer. Elle fera appelle au composant 5 Vérificateur de bloc pour l'insertion et la vérification de sa conformité. Et elle génère une exception si le hash du bloc précédent n'est pas bon.

Signature : void insert (Bloc b)

4.3- Recherche d'un bloc

Cette fonction recherche un bloc dans la blockchain et renvoie le bloc trouvé. Sinon elle renvoie une erreur. Elle prendra en paramètre l'identifiant du bloc cherché ou le hash du bloc. Génère une exception si le bloc recherché n'existe pas.

Signature : Bloc findByHash (std :: string hash)
Signature : Bloc readByIndex (int index)

4.4- Vérification de tous les blocs

Cette fonction utilise la fonction de lecture du fichier pour identifier tous les blocs puis utilisera le composant 5 Vérificateur de bloc pour valider les blocs identifiés. Si le fichier n'est pas conforme elle renverra un message d'erreur.

Signature : bool verification()

4.5- Lecture de tous les blocs

Cette fonction lit tous les blocs, et les renvoie sous forme de liste. En amont elle fait appelle à la vérification de tous les blocs puis elle renvoie la liste de tous les blocs présents dans la blockchain.

Signature : std :: vector<Bloc> readAll()

4.6- Lecture d'un bloc sous forme de string

Cette fonction renvoie une chaine de caractère représentant le bloc sous son format JSON. Elle prendra en paramètre un booléen. Si l'argument a pour valeur true la fonction renverra la chaine de caractère du bloc avec le hash. Sinon elle renverra la chaine sans le hash et nonce. Le second argument est soit le hash du bloc que l'on souhaite retourner sous forme de string, soit sa position dans la chaine.

Signature : String ToString (bool allInfos,, std :: string hash)
String ToString (bool allInfos,, int index)

II- Test

1- Cas de test numéro 1 – ouverture d'un fichier vide

Ce code va tester la lecture du fichier représentant la blockchain mais ne comportant aucun bloc. Et nous vérifierons donc que la liste renvoyée en retour est de taille 0.

2- Cas de test numéro 2 – insertion d'un bloc

Ce code va tester l'ajout d'un nouveau bloc dans la blockchain et donc dans le fichier. Nous allons donc appeler la fonction d'insertion d'un bloc deux fois (pour deux blocs différents). Puis nous allons à nouveau ouvrir le fichier et vérifier que désormais la taille de la liste renvoyée est de longueur 2.

3- Cas de test numéro 3 – recherche d'un bloc

Nous allons ensuite tester que lorsque l'on appelle la fonction de recherche d'un bloc en passant en paramètre le hash du premier bloc que nous venons d'ajouter, la fonction nous renvoie bien un bloc identique à celui que nous avons insérer dans la blockchain. Nous allons ensuite reproduire le même test mais en appelant cette fois la fonction avec comme paramètre le chiffre 2. Et nous allons vérifier que le bloc retourné est bien identique au bloc que nous avons inséré précédemment.

4- Cas de test numéro 4 – vérification de tous les blocs

Dans ce code nous testons que l'appelle de la méthode de vérification de la blockchain nous renvoie bien « True » pour le fichier que nous avons construit. Nous testons ensuite cette méthode avec une autre blockchain chargée d'un fichier contenant déjà des blocs mais n'étant pas conformes. Et l'on vérifie que la méthode de vérification nous renvoie bien une erreur.

5- Cas de test numéro 5 – lecture de tous les blocs

Ici nous allons appeler la fonction de lecture des blocs pour la blockchain que nous avons fabriqué depuis le début de ces tests. Et nous allons vérifier qu'elle nous renvoie bien une liste de deux blocs étant égaux à ceux que nous avons ajoutés précédemment.

6- Cas de test numéro 6 – lecture en to String

Pour finir nous allons tester que lorsque nous lisons un bloc, la fonction de lecture en to String nous renvoie bien ses informations mais sous forme de liste de caractère.