Composant 3 : blockchain

Auteurs

Louis Ledoux

Théo Chennebault

Nikola Spaci

Tom Moore

Do Thanh Long LE

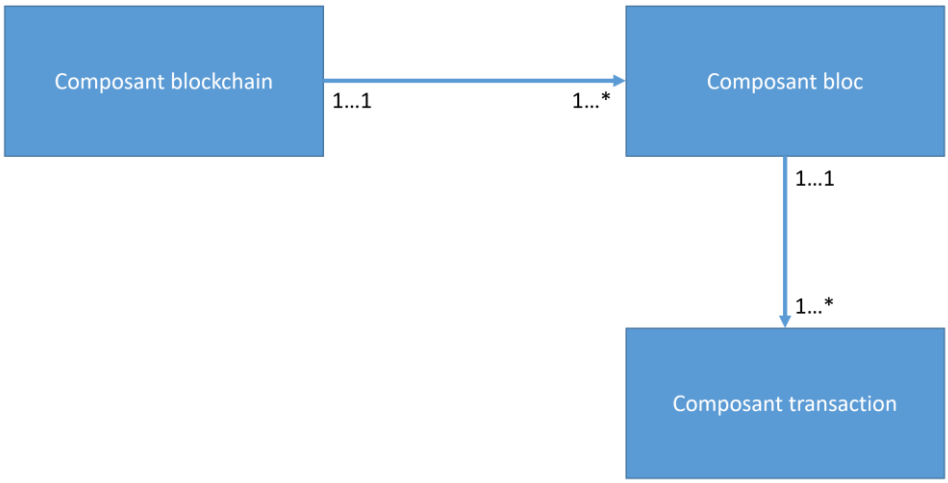
Version :

1.1.1

Description :

La blockchain est une liste chainée ordonnée de blocs. Ainsi, le composant blockchain encapsule le composant bloc. Il permet également de sauvegarder la blockchain au format Json. Enfin, il doit pouvoir charger la blockchain à partir d'un fichier Json. Il est à noter que le composant bloc encapsule le composant transaction. Ainsi, lors du mapping au format Json, une dépendance à ce composant est ajoutée.

Schéma de dépendances aux autres composants :



Il faut donc s'assurer que le composant bloc soit bien à jour au sein du répertoire.

Signature méthode Python :

Blockchain(json) : création de l'ensemble de la blockchain à partir d'un fichier Json contenant la blockchain.

Void from_json(json) : Déserialisation d'un fichier JSON contenant une liste chainée de bloc.

Json to_json() : Retranscrit l'ensemble de la structure blockchain (ensemble des objets blocs de liste chainée) au format JSON.

Int getNbBloc() : Retourne un entier correspondant au nombre de bloc dans la liste chainéé de la blockchain.

Bloc getBloc(index) : Récupère le bloc à la position i de la blockchain.

Void addBloc(bloc) : Ajoute un bloc dans la liste blockchain si le hash précédent de ce bloc est bien égal au hash du dernier bloc de la liste (appelle la méthode verifyPreviousBlock).

bool verifyPreviousBlock(Bloc b, Bloc bPrev) : Fonction qui prend en paramètre, un bloc i et le bloc qui le précède (i-1) dans la blockchain. Cette fonction retourne un booléen indiquant si l'attribut previous hash du bloc i correspond au hash du bloc i-1.

Tests :

Pour tester le composant nous avons créé un fichier de test nommé "test_blockchain.py" qui contient ensemble de fonctions tests.

testCreateBlockchainFromJson(data,blockchain) : avec data le mock blockchain désérialisé et blockchain l'objet Blockchain instancié.

Les premiers tests réalisés consistent à s'assurer que les données issues d'un objet blockchain sérialiser au format JSON peut être désérialisé et stocké au sein d'un objet blockchain (ensemble des blocs). Il faut s'assurer que chaque attribut du bloc soit bien chargé, même quand l'attribut est associé à une autre classe. Par exemple comme c'est le cas pour les champs tx0 qui a un TXM ainsi que pour la liste de TX (txs). Pour s'assurer que le chargement est opérationnel, on réalise un mock en JSON d'une blockchain. C'est ce mock qui sera utilisé dans la phase de test. On s'assure ensuite que la sérialisation est correcte en sauvegardant les différents blocs de la blockchain dans un JSON, on s'assurera que ce nouveau JSON est identique au Mock JSON précédemment utilisé.

testNBBloc(data,blockchain) : avec data le mock blockchain désérialisé et blockchain l'objet Blockchain instancié.

Le test consiste à s'assurer que le nombre de bloc présent dans l'object blockchain est bien identique au nombre de bloc présent dans le Mock JSON d'une blockchain.

testGetBloc(data,blockchain) : avec data le mock blockchain désérialisé et blockchain l'objet Blockchain instancié.

Le test consiste à s'assurer que le premier bloc de l'objet blockchain est identique au premier block du Mock Json.

testVerifPreviousHashAndAddCorrectBloc(blockchain) : avec blockchain l'objet Blockchain instancié.

Le test consiste à vérifier deux méthodes que nous avons développés :

- La méthode qui s'assure de la validité du previous hash d'un bloc
- La méthode qui s'assure de l'ajout d'un bloc en fin de liste chainée

Pour ce test nous allons partir d'un bloc JSON correct c'est à dire qui a un previous hash correct. Afin de s'assurer que la méthode verifyPreviousBlock() retourne True et que l'ajout en fin de liste est réalisé.

testVerifPreviousHashAndAddIncorrectBloc(blockchain) : avec blockchain l'objet Blockchain instancié.

Le test consiste à vérifier deux méthodes que nous avons développés :

- La méthode qui s'assure de la validité du previous hash d'un bloc
- La méthode qui s'assure de l'ajout d'un bloc en fin de liste chainée

Pour ce test nous allons partir d'un bloc JSON incorrect c'est à dire qui a un previous hash incorrect. Afin de s'assurer que la méthode verifyPreviousBlock() retourne False et que l'ajout en fin de liste n'est pas réalisé.