Projet Programmation par Composants

Document de spécification

Composant 5 – Vérificateur de Bloc

Auteurs :

AMRANI Said
RAVOAVY HARIMAMPIANINA Christian Mahery
NGUYEN Michel

M2 MIAGE IF APP

Historique des versions			
Version	Date	Auteurs	Modifications
1.0	19/04/2020	Said, Christian, Michel	Création de la première version du document
2.0	26/04/2020	Said, Christian, Michel	Mise à jour après discussion

Table des matières

Description

Contexte

La blockchain est une technologie de stockage et de transmission d'informations, transparente, sécurisée, et fonctionnant sans organe central de contrôle (définition de Blockchain France). C'est une utilisation astucieuse de deux technologies : la cryptographie et les réseaux peer to peer.

Par extension, une blockchain constitue une base de données qui contient l'historique de tous les échanges effectués entre ses utilisateurs depuis sa création

Le projet consiste à construire une petite blockchain (sans la diffusion peer to peer)

<u>Schéma de fonctionnement d'une blockchain :</u>

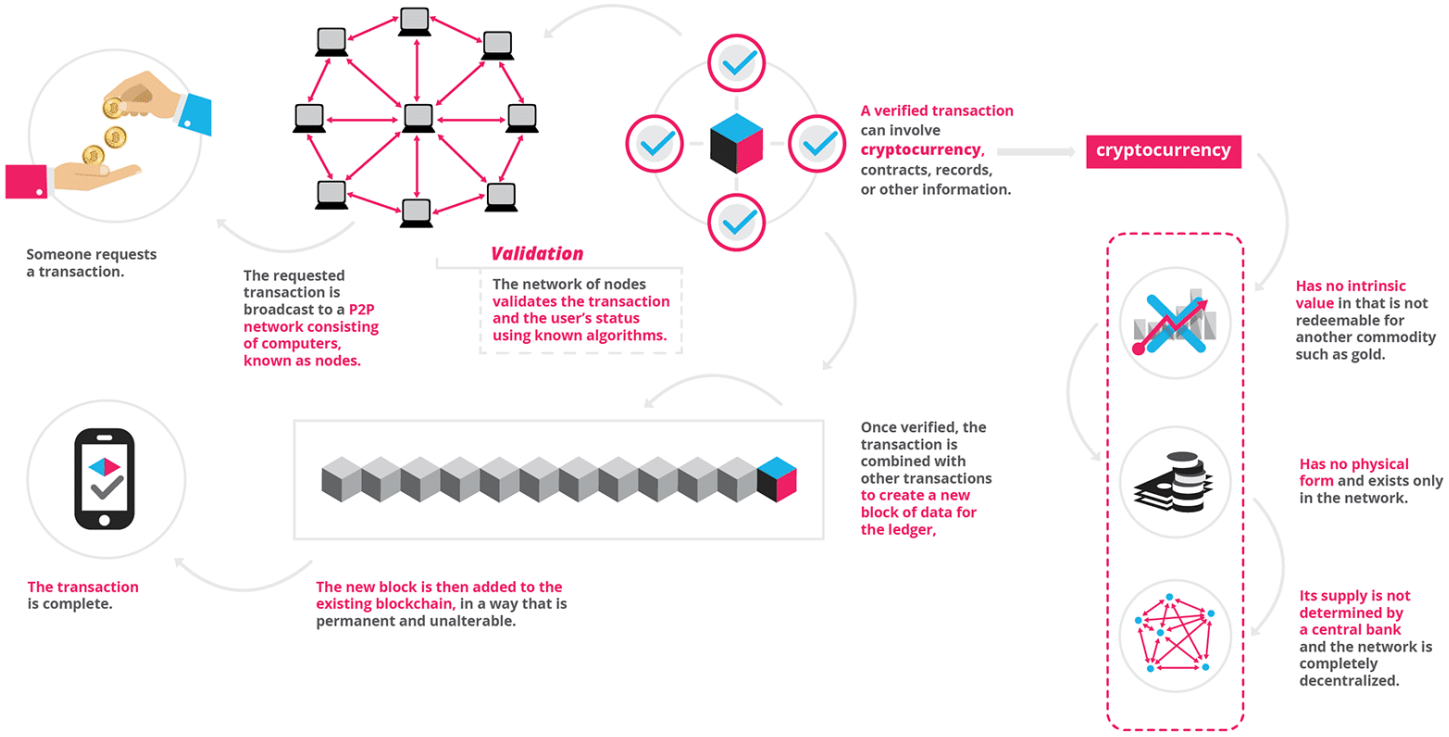


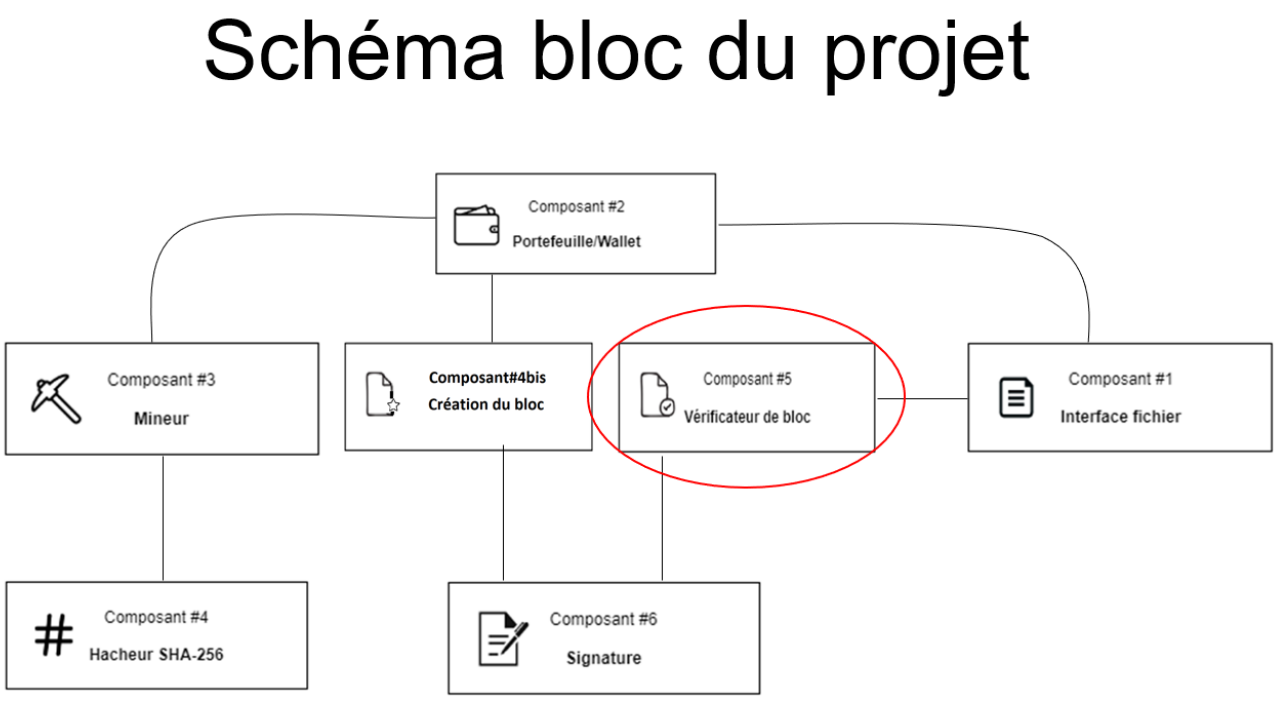
Schéma bloc des composants connexes

Le projet de programmation est divisé en 6 composants distincts :

1. L'interface Fichier
2. Le portefeuille (ou wallet)
3. Le mineur
4. Le Hacher et le Créateur de bloc
5. Le vérificateur de bloc
6. La signature

Dans ce document, nous allons nous concentrer sur le 5ème composant, le vérificateur de blocs, en expliquant comment il fonctionne et comment il devra interagir avec les autres composants. Nous mettrons aussi en place une stratégie de tests unitaires rédigés en Python.

Voici un schéma qui récapitule la disposition des composants.



Interface et interaction avec chaque autre composant

Notre composant utilisera les interfaces créées par les autres groupes, pour obtenir les transactions, les blocs, et vérifier leurs validités.

Il faudra ainsi déterminer si un bloc est bon ou non, et s'il l'est, alors on l'ajoutera à la chaine : d'où le nom de blockchain.

Notre composant (le 5ème), interagit avec le premier et le sixième composant.

On recevra le fichier du 1er composant, qui représentera la blockchain. On aura dedans une liste de blocs validés. Nous devrons vérifier, dans une boucle, si chaque bloc est bien conforme. On retournera ainsi un booléen qui indique si le fichier est valide, c'est-à-dire si tous les blocs le sont.

Après avoir vérifié les transactions, et validé ou non la composition du bloc, on transmettra le résultat au groupe qui se charge de la signature du bloc.

Déclarations de fonctions python d'interface et leurs arguments

La classe VerificateurDeBloc contient la méthode publique suivante :

Def VerificateurBloc(bloc) prend comme argument un bloc de type Bloc et a comme valeur de retour un booléen. La valeur du booléen sera True si le bloc est conforme, c'est-à-dire toutes les fonctions intermédiaires retournent toutes True et False sinon.

Entre autres, cette méthode publique appelle des fonctions intermédiaires suivantes qui seront privées. Celles-ci se trouvent en annexe.

Cas d'erreurs

Les erreurs qui peuvent se produire au moment des appels des fonctions :

- bad exception : Lancée si aucun catch ne correspond à un objet lancé.
- invalid_argument : Argument invalide passé à une fonction.
- out_of_range : Erreur d'indice de tableau, notamment pour le test du hash.
- bad_type_id : Lancée s'il se produit une erreur lors d'un typeid.

Tests

Cette partie sera assurer en utilisant les tests unitaires pour s'assurer du bon fonctionnement de nos méthodes

'L'objectif d'un test unitaire est de permettre au développeur de s'assurer qu'une unité de code ne comporte pas d'erreur de programmation. C'est un test, donc les vérifications sont faites en exécutant une petite partie (une « unité ») de code. En programmation orientée objet, l'unité est la classe[1]. Un test est donc un programme qui exécute le code d'une classe pour s'assurer que celle-ci est correcte, c'est-à-dire que ses résultats correspondent à ce qui est attendu dans des assertions prédéfinies (ex : Addition(1,2) = 1 + 2 doit être vrai).

Concrètement, un test, c'est du code. À chaque classe d'une application, on associe une autre classe qui la teste'

Concrètement, il s'agit de créer les méthodes de test suivantes qui testeront les méthodes d'interfaces et leurs arguments :

Test_VerificateurBloc()

Pour cette méthode on passe en paramètre de la méthode **VerificateurBloc** pour un premier temps un bloc correspondant pour voir s'il retourne un True et pour un deuxième temps un bloc non correspondant pour voir s'il retourne un False. Si ça ne retourne pas les résultats attendus alors il y a une erreur dans la méthode.

Les tests des fonctions intermédiaires se trouvent en annexe.

Annexe

Python

def VerificateurHash(hash[HASH_SIZE]): prend comme argument un tableau de caractère de taille HASH_SIZE et a comme valeur de retour un booléen. La valeur du booléen sera True si le Hash est conforme en appelant la fonction calculant l'hash du composant HASH sur le bloc précédent, si cela correspond bien, True sera retournée et False sinon.

def VerificateurTransaction(transaction): prend comme argument une transaction de type TX et a comme valeur de retour un booléen. La valeur du booléen sera True si la transaction est conforme, si oui, True et False sinon. Cette fonction appellera la fonction VerifierSignature du composant 6.

Def VerificateurNonce(nonce): prend comme argument un nonce de type entier non signé et a comme valeur de retour un booléen. La valeur du booléen sera True si le nonce est conforme, c'est-à-dire unique pour toute la chaine, et False sinon.

Def VerificateurNumBloc(num): prend comme argument un num de type entier et a pour valeur de retourne un booléen. La valeur du booléen sera True si le numéro est conforme, c'est-à-dire unique pour toute la chaine, False sinon.

Def VerificateurGainMineur(gainMineur): prend comme argument un gainMineur de type TXM et retourne un booléen. La valeur du booléen sera True si le gainMineur est conforme et False sinon.

Transaction du mineur :

- Une seule transaction de ce type par block
- Pas d'entrée
- Montant fixe: 10
- Clé publique du mineur

Test

Test_ VerificateurHash()

Cette méthode de test teste la fonction **VerificateurHash en** passant en argument un tableau de caractère de différente taille et de faire une sorte que le hash calculer en appelant la fonction de Hash sur le bloc soit dans le premier cas conforme et dans le deuxième cas non conforme et voir si on a bien le bon retour de la fonction

Test_ VerificateurTransaction()

On teste la méthode VerificateurTransaction en passant en paramètre une transaction conforme et une transaction non conforme pour voir si on aura bien un TRUE pour la première transaction puis un false pour la deuxième transaction, dans le cas contraire, il y a forcément une erreur dans la méthode

Test_ VerificateurNonce()

Cette méthode teste **VerificateurNonce** en passant en paramètre un Nonce non conforme et voire si ça retourne un false et un Nonce conforme pour vérifier c'est ça retourne un True

Test_ VerificateurNumBloc()

On teste la méthode **VerificateurNumBloc en** passant en paramètre un entier qui n'est pas le même pour toute la chaine pour vérifier si ça retourne un False et un entier unique pour toute la chaine pour tester si ça retourne un True

Test_ VerificateurGainMineur ()

Teste la méthode **VerificateurGainMineur** en passant un gain mineur conforme et un autre non conforme pour vérifier si on obtient un True et False respectivement

En fin, on teste s'il y a bien une bonne communication entre le composant 1 et 6 et tester le résultat final.

Sources

https://blockgeeks.com/guides/fr/quest-ce-que-la-technologie-blockchain-un-guide-pas-a-pas-pour-debutants

https://blockchainfrance.net/decouvrir-la-blockchain/c-est-quoi-la-blockchain/

https://fr.wikibooks.org/wiki/Introduction_au_test_logiciel/Tests_unitaires