Rapport Projet Cassiopée n° 81

Étudiants : Nathanaël Denis et Pierre Chaffardon
Encadrantes : Maryline Laurent et Nesrine Kaaniche

Sujet: Système de vote pour le contrôle d'accès, basé sur la technologie blockchain.

Sommaire

I. Préliminaires

1) Motivation

Le choix du sujet s'est effectué assez naturellement pour nous, étant donné que celui-ci porte sur une problématique de sécurité évidente, faisant ainsi écho à la voie d'approfondissement que nous avons tous les deux choisie (Sécurité des Systèmes et des Réseaux). Les problématiques de contrôle d'accès ont toujours été au coeur des enjeux de sécurité informatique, et le point de vue proposé dans ce travail pour les traiter nous a très vite intéressés .

L'utilisation de la technologie blockchain, traditionnellement connue pour son utilisation dans le cadre des transactions de crypto-monnaies telles que le Bitcoin, s'est révélée particulièrement appropriée pour ce type de solution. Utilisée par la fondation Ethereum puis dans de nombreux types de transactions au sein d'entreprises variées avec la mise en place de "smart contracts", il semble que cette technologie continue de présenter un intérêt non négligeable dans le domaine de la sécurité, entre autres, aujourd'hui comme dans les dans les années à venir.

2) Rôle de la blockchain

Développée depuis une vingtaine d'années, la technologie Blockchain permet le stockage et la transmission d'information sans organe de contrôle, c'est-à-dire de manière décentralisée. Il s'agit, de manière imagée, d'un grand livre de compte distribué dans lequel sont stockées sous forme de blocs toutes les transactions ayant eu lieu depuis le démarrage du système en question.

L'objectif est de s'assurer que tous les noeuds du réseau sont capables de fonctionner de concert et de produire un consensus, en particulier pour l'écriture de nouveaux blocs. Bien qu'il existe un certain nombre de "preuves" permettant à un noeud d'écrire un nouveau bloc, les deux qui demeurent les plus utilisées sont la "preuve de travail" et la "preuve de participation".

La première se base sur un problème mathématique qui permet de vérifier que le mineur qui s'apprête à écrire dans le registre a bien réalisé un travail spécifique. Traditionnellement (ou du moins pour ce qui est du bitcoin), le travail en question fait intervenir un challenge cryptographique résolvable par force brute et qui demande une puissance de calcul très importante, sachant en outre que seul le premier mineur qui réussit le challenge se trouve récompensé (cela permet de

stimuler les mineurs afin de fournir au système une puissance de calcul suffisante). La difficulté du problème augmente en fonction du nombre de mineurs, de sorte que le protocole soit inviolable sauf possession de plus de la moitié de la puissance de calcul disponible, ce qui demeure extrêmement compliqué compte tenu des ressources à déployer. Ethereum a d'ailleurs subi une attaque de ce genre (dite "attaque des 51%") cette année.

La seconde consiste à demander aux utilisateurs de prouver qu'ils possèdent une certaine quantité de crypto-monnaie par exemple afin de vérifier leur participation au système. Le système Peercoin fut le premier à mettre ce type de consensus à l'oeuvre. Notons par ailleurs que dans son développement final Ethereum utilise un mélange de la preuve de travail et de la preuve de participation.

Il existe de surcroît plusieurs modes de gouvernance possibles, à savoir public, semi-privé et privé. Dans le premier cas tout le monde peut lire ce qu'il y a dans la blockchain et potentiellement y écrire. Pour des réseaux semi-privés la blockchain est accessible en lecture mais l'accès est restreint en écriture (souvent un organe central possède le droit d'écriture). Enfin pour le dernier cas le processus n'a pas besoin d'être public mais nécessite une sécurité accrue, l'intérêt majeur résidant dans la robustesse du système et la traçabilité.

L'aspect démocratique de la blockchain présente également un intérêt marqué pour certains types de solution. Bien que le principe de vote électronique à grande échelle ait été sujet à de nombreux débats récemment, tant en termes de fiabilité qu'en coût énergétique, il n'en reste pas moins que pour des environnements privés d'envergure faible, ou moyenne, ce système se puisse se révéler adapté.

3) Choix des outils utilisés

Nous avions initialement le choix entre 2 projets open sources pour développer cette solution : Ethereum et Hyperledger. Le premier a déjà fait ses preuves et jouit d'une renommé certaine concernant le développement de blockchain mais se trouve aussi très centré sur les crypto-monnaies, tandis que le second, poussé par la communauté Linux et Red Hat, permet de travailler sur une logique de transaction plus générale et propose des frameworks facilitant la partie développement.

Nous avons donc fait le choix d'utiliser le framework Hyperledger Composer, en suivant notamment un MOOC sur la plateforme edX intitulé Blockchain for Business – An Introduction to Hyperledger Technologies. Celui-ci présente Hyperledger de manière générale ainsi que le concept de blockchain privée, avant d'aider à la mise en place des différents outils possibles selon l'objectif fixé.

A visée plus générale que les autres frameworks Hyperledger (Sawtooth ou Iroha par exemple), Hyperledger Composer nous permet d'implanter notre solution de contrôle d'accès et de modification de documents et de la tester de manière assez fluide, ce dont nous reparlerons plus en détails par la suite dans la partie dédiée.

II. Implémentation de la solution

Concrètement, la solution s'implémente dans différents fichiers. La partie modélisation d'une part s'est faite dans un fichier à part, où l'on a défini les différents participants (utilisateurs et clients), ainsi que les "assets" (documents), les "events" et les "transactions" (modifications des documents). Un autre fichier a été dédié de même au développement de la logique de transaction en utilisant le langage Javascript, où l'on implémente par exemple les fonctions de modification de documents et d'écriture des events. Les permissions de base sont gérées dans un fichier spécifique, comprenant par exemple les droits de lecture et de modification. Un dernier fichier est dédié aux requêtes appelées par certaines fonctions implémentées.

1) Modélisation

Les différentes entités entrant en jeu sont définies dans un fichier nommé org.vote.cto.

On considère ici deux types d'intervenants appelés "participants" : les clients et les utilisateurs. Les clients mettent en ligne des documents, tandis que les utilisateurs les lisent, voire les modifient. Tous deux sont identifiés de manière unique par un attribut nommé clientId ou utilisateurId, chaque utilisateur possédant en outre une liste de documents sur lesquels il a un droit de lecture.

Remarquons qu'ici les attributs servant à identifier les participants sont sous format "string" et non "integer" comme cela se fait traditionnellement.

Ici les "assets", c'est-à-dire les ressources concrètes manipulées, sont des documents, possédés par un client spécifique, identifiés de manière uniques par un attribut nommé documentId, et constitués d'une chaîne de caractère d'une certaine longueur. Notons bien qu'ici l'utilisation de chaînes de caractères permet quand même l'utilisation d'images ou de tableaux, prenant ainsi en charge les formats les plus classiques dans le monde de l'entreprise.

On définit de même deux "events" qui correspondent à ce qui va être écrit dans le registre de la blockchain. Le premier gère la modification d'un document (avec les parties potentiellement insérées et/ou supprimées) ainsi que le résultat du vote correspondant (protocole appelé et résultat). Le second correspond à l'ajout d'un utilisateur à la liste des contributeurs d'un document.

Enfin on implémente deux "transactions" qui correspondent à la modification de l'état du système, que ce soit au niveau des participants ou des assets, autrement dit des ressources. Chacune d'elle est suivie par l'event approprié, qui détaille les événements survenus. Il y en a donc une qui sert à ajouter un nouveau contributeur à un document et l'autre qui démarre les votes (avec le protocole choisi) et qui effectue les modifications sur les document selon le résultat du vote en amont.

Pour finir on définit un type "enum" qui correspond au protocole de vote, qui peut donc soit se faire à l'unanimité soit à la majorité.

2) Développement de la logique de transaction

La logique de transaction, constituée des fonctions de modification de documents et de la mise en oeuvre des protocoles de vote, est définie dans un fichier nommé logic.js et faisant appel au langage javascript.

Le problème qui se pose ici consiste à gérer le contrôle d'accès sur des documents partagés par des clients au niveau des droits en lecture et en écriture. Il s'agit donc d'une solution à visée collaborative implémentée dans un environnement privé, correspondant par exemple à une entreprise, ou un laboratoire de recherche, d'au moins une dizaine de personnes.

Lorsqu'un utilisateur possédant le droit de lecture (et d'écriture sous-jacent) sur un document et souhaite effectuer une modification, il la soumet à un vote qui fait appel à la liste des contributeurs dudit document et du client.

Au lancement du vote, l'utilisateur définit un timer qui donne la durée du vote. Une fois que le timer arrive à échéance, l'event concernant la modification (et le vote) est publié, et selon que la modification ait été ratifiée ou non, elle est appliquée -ou pas- au document. Dans notre version en local, il n'est pas possible d'implémenter un timer car la demande de réponse se fait via la méthode prompt() qui est bloquante et attends une réponse de l'utilisateur. Cela est logique, car si la fenêtre a été ouverte, l'utilisateur est très probablement devant son écran et il n'y a pas lieu d'attendre.

Par ailleurs nous avions prévu d'envoyer des mails pour prévenir les contributeurs qu'une modification doit être validée ou refusée. Cela n'est pas possible avec Javascript. En effet, l'exécution du code Javascript se fait en local, et avec des attaques de type Cross-site scripting, les machines des utilisateurs pourraient facilement être utilisées pour des campagnes de spam.

On choisit arbitrairement l'utilisateur ayant requis le vote pour écrire dans le registre, faisant ainsi abstraction des problèmes liés au consensus (proof of work, proof of elapsed time etc) et à la récompense rencontrés traditionnellement dans les blockchains liées aux crypto monnaies (Bitcoin, Ethereum etc). Le système évoluant dans un environnement privé, il paraît plus cohérent de ne pas s'encombrer avec un processus de consensus trop gourmand en temps et en énergie lorsque tous les utilisateurs sont de confiance et prêts à donner un peu de leur puissance de calcul.

Enfin concernant l'aspect conflictuel que l'on peut rencontrer sur des documents modifiés en parallèle, les votes se déroulant l'un après l'autre, chaque utilisateur est en possession de la dernière version du document avant une quelconque demande de modification. Ainsi on évite ce problème récurrent rencontré lors de la majorité des travaux collaboratifs.

3) Autorisations

L'ensemble des autorisations sur les ressources du réseau est défini dans le fichier permissions.acl, qui contient un ensemble de règles spécifiques comme plus générales.

Concrètement les trois principaux types d'autorisations explicités dans ce fichier sont CREATE (création ainsi que modification), READ (lecture) et DELETE (destruction).

Les premières règles concernent l'administrateur du réseau déployé, lui attribuant basiquement tous les droits sur toutes les ressources disponibles (assets, participants, transactions etc).

On spécifie de même par exemple que tous les utilisateurs ont le droit de lecture et de modification sur les documents qui sont présents dans leur liste de documents modifiables. On peut également s'attarder sur un utilisateur en particulier lui attribuant un rôle spécial en précisant des droits spécifiques concernant certains documents. On gère en outre dans cette partie les droits des clients qui mettent en ligne des documents, sur lesdits documents, à savoir création/lecture/destruction.

Il existe d'autres éléments du système sur lesquels il faut établir des droits d'accès. Notamment, il faut que les utilisateurs puissent écrire de nouvelles transactions dans la blockchain, et doivent donc pouvoir aussi modifier l'historique des transactions.

La définition de telles règles demeure est nécessaire puisque c'est l'utilisateur qui a soumis une modification à un vote qui écrit le nouveau bloc. Il était possible également de laisser l'administrateur du système écrire le bloc, ce qui est seulement viable quand le système est petit, mais devient moins pertinent dès que le taille du système augmente. Notons également que sur le principe, la blockchain vise à définir des systèmes décentralisés, il semble donc plus pertinent de mettre en place un système d'écriture décentralisé .

Hyperledger Composer prévoit donc un rôle d'administrateur qui est nécessaire au moins au début pour définir les utilisateurs du système et les ressources initiales. Il est par conséquent possible, à terme, de s'émanciper de cet administrateur en utilisant uniquement des transactions, qu'il faut néanmoins programmer. Dans un système d'importance relative, garder un rôle administrateur paraît plus sécurisé néanmoins pour la pérennité du réseau, ce qui ne serait pas forcément le cas pour un système plus petit.

4) Requêtes

Hyperledger Composer intègre un langage dédié pour faire des requêtes sur tous les objets du système: utilisateurs, ressources etc. A la manière du langage SQL pour les bases de données, il est possible de sélectionner des objets selon des conditions précises, comme par exemple l'ensemble des contributeurs qui peuvent écrire dans un document spécifique. La condition est écrite en langage Javascript, et il est possible d'utiliser les méthodes Javascript, telle que la présence d'un objet dans un tableau.

La requête est composée de deux champs, *description* qui explique le rôle de la requête et *statement* qui constitue la requête en elle même.

L'intérêt de ces requêtes est de faire une sélection d'objets particuliers sur lesquels le développeur souhaite travailler dans le fichier logic.js. Nous n'avons pas eu besoin d'utiliser les requêtes dans cette optique.

III. Déploiement de la solution

1) Influence du framework HyperLedger Composer

L'utilisation de Hyperledger Composer a influencé la programmation de manière significative. L'implémentation des Smart Contracts comme spécifié dans le cahier des charges initial est géré de manière autonome par le framework. Le Smart Contract pour les droits d'accès y est directement intégrée, via une Access Control List (ACL) classique. Le contrat appelé pour le gérer le vote est programmé dans le fichier qui gère la logique de transaction (logic.js). Les deux Smart Contracts ne sont donc pas conçus ni utilisés de la même manière.

Par ailleurs, Hyperledger Composer offre deux manières de programmer. Soit directement en ligne de commande avec un système d'exploitation Linux, soit dans l'interface utilisateur prévue, Composer Playground. Comme nous suivions un MOOC sur la plateforme edX, nous avons choisi la première méthode. Nous avons programmé les fichiers de modélisation et de transactions (respectivement le seul fichier avec l'extension .cto et le fichier logic.js) dans un éditeur de texte puis nous avons déployé le réseau une première fois avec les lignes de commande.

Une fois le réseau déployé, la correction des erreurs s'est faite dans l'interface utilisateur, qui permet de diagnostiquer quelques erreurs, bien que le débogage soit extrêmement rudimentaire (peu d'indications données, des difficultés à trouver les portions de code qui correspondent aux erreurs etc).

La nature différente des deux Smart Contracts permet de résoudre un problème qui paraissait critique au départ : l'interdépendance des deux Smart Contracts. En effet, le protocole de vote nécessite de connaître les droits d'accès aux documents qui sont contenus dans l'autre Smart Contract. Il est possible d'accéder à l'ACL depuis le fichier logic.js et donc de connaître les droits sur les documents pour chaque utilisateur au moment du vote.

Le choix de Composer nous a forcé à choisir des langages de programmation spécifiques, notamment le langage de modélisation propre à ce framework. L'utilisation du Javascript pour la gestion des transactions impose l'utilisation d'objets particuliers (registres, fonctions asynchrones) que nous n'avions pas l'habitude d'utiliser.

Comme Hyperledger offre de nombreux outils spécialisés et assez récents (le projet a été initié en 2015), il est particulièrement difficile de trouver l'origine de ses erreurs et la correction des problèmes mêmes mineurs est particulièrement longue.

Par ailleurs, les formations sont basiques et expliquent le fonctionnement de Hyperledger Composer dans les grandes lignes, mais les problèmes profonds - de compréhension de l'outil notamment - sont très compliqués à résoudre. Néanmoins, le choix d'un autre outil aurait probablement mené au même problème y compris pour Ethereum bien qu'il soit un peu plus ancien (2013).

2) Test de la solution

Le test de la solution se fait à partir de l'interface graphique proposée par Comper Playground. Il est nécessaire de charger les différents fichiers requis pour l'élaboration d'une blockchain sous Hyperledger (le fichier de modélisation, celui gérant la logique de transaction, le fichier des permissions et celui des requêtes).

Il convient ensuite de déployer le réseau afin d'être sûr de sa cohérence et de traquer les éventuelles erreurs (comme expliqué précédemment), afin de finalement tester la solution. Pour se faire il faut instancier des documents ainsi que des clients et des utilisateurs, ayant certains droits sur lesdits documents.

L'interface dédiée au test permet ensuite de faire des transactions, en renseignant les informations correspondantes définies dans le fichier de modélisation (identifiants de l'utilisateur et du document pour une modification

typiquement). Concernant le protocole de vote, celui-ci se met en marche et demande effectivement aux utilisateurs signifiés dans la transaction si OUI ou NON ils valident la modification dans une petite fenêtre dédiée à cet effet.

Un onglet spécifique de l'interface permet également d'avoir accès au registre de la blockchain et ainsi constater les modifications demandées et potentiellement effectuées depuis le lancement du système sur l'ensemble des documents. On trouve alors les différents blocs contenant les transactions (en l'occurrence un bloc par transaction) de même que les différents events associés avec le système classique de hash (qui correspond ici à l'ID de l'event).

Il est également possible lorsque l'on est connecté en tant qu'administrateur en local de s'identifier à la place d'un utilisateur donné afin de tester les problèmes de contrôle d'accès. Ainsi il est possible de détecter d'éventuelles permissions abusives ou au contraire des absences d'autorisations suspectes.

IV. Exemple d'utilisation

Nous allons présenter dans cette dernière partie un cas d'utilisation de notre solution, avec deux documents emblématiques nommés Bible et Codex Gigas. Les deux clients qui rentrent en jeu ont donc pour identifiant Dieu et Diable, et les autres utilisateurs ont pour identifiant des noms d'apôtres et de démons.

Concernant les autorisations attribuées, les utilisateurs avec un identifiant d'apôtre ont le droit de lecture et de modification sur la Bible mais aucun droit sur le Codex Gigas, et vice-versa pour les utilisateurs avec un nom de démon.

Lorsqu'un apôtre souhaite modifier la Bible grâce à la blockchain, il demande aux autres apôtres s'ils sont d'accords pour la modification. Si le résultat du vote est positif, l'apôtre peut modifier le document et écrit la transaction dans la blockchain. En cas de refus, la modification n'est pas actée. Il en est de même avec les démons et le Codex Gigas.

C'est ce cas d'utilisation que nous avons implémenté sur nos machines avec Hyperledger Composer et qui est l'objet principal de nos démonstrations pour le moment. Néanmoins on peut envisager de nombreux autres cas d'utilisation plus concrets où le partage de documents peut être pertinent.

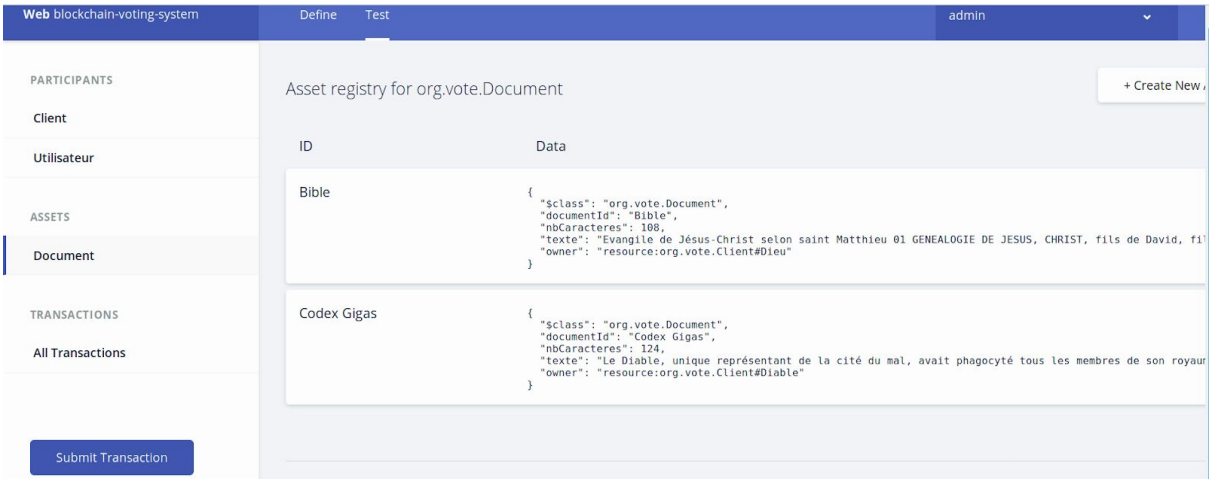
Une encyclopédie collaborative pourrait typiquement employer cette méthode pour faire valider en amont par les contributeurs à l'article une modification. Plutôt que de corriger *a posteriori*, et donc de propager momentanément des fausses informations, les articles seraient validés en amont par la communauté. En plus d'étendre potentiellement le volume d'information, cette méthode accroîtrait leur fiabilité et ce de manière permanente.

Des documents sensibles comme certains textes de loi - dans l'optique où ces derniers seraient numérisés - pourraient être soumis à un vote à une étape particulière du processus de création d'une loi. On ferait alors appel aux sénateurs qui ont pour rôle justement de voter les lois, ou encore au conseil constitutionnel qui vérifie que les nouvelles lois sont en accord avec la constitution, qui est du point de vue juridique la norme suprême.

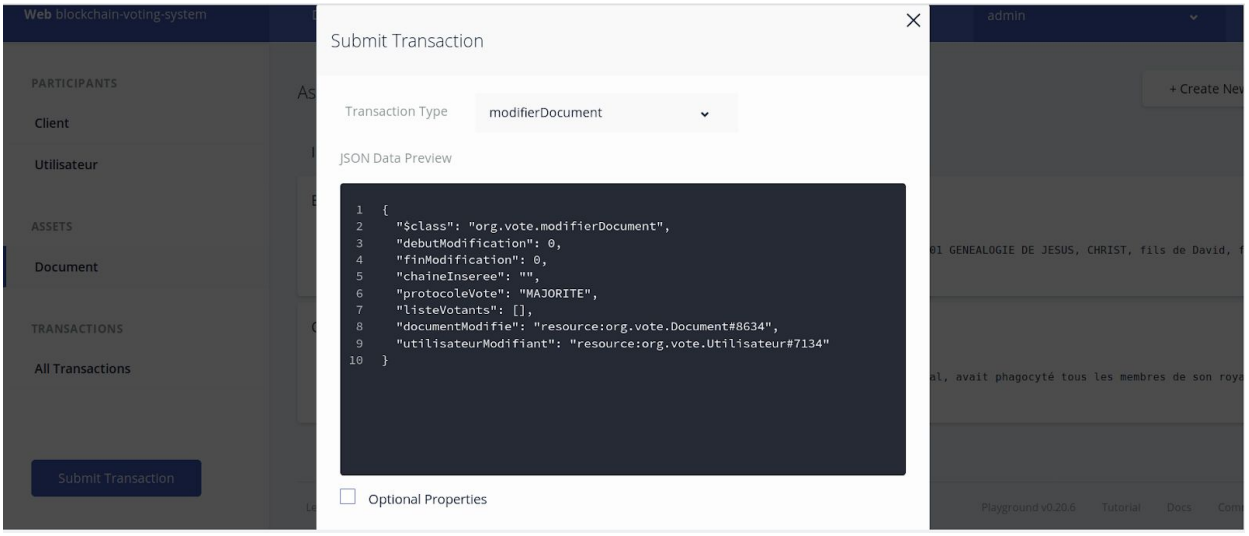
Dans tous les cas, il convient de discuter du coût et de la pertinence de la technologie blockchain avant de penser à son déploiement. Ce système est bien évidemment vulnérable aux attaques extérieures. Hors, rendre le mode de vote moins fiable peut s'avérer très dangereux pour un pays. Il est fort probable que des puissances étrangères tentent d'interférer avec l'établissement des lois d'un pays avec leur arsenal numérique si jamais les votes venaient à être dématérialisés.

A noter que par opposition aux crypto-monnaies où les utilisateurs sont dissuadés d'attaquer car mettre en péril le système revient à prendre le risque de faire chuter la valeur des crypto-actifs détournés, d'autres systèmes seraient au contraire intéressants à détruire pour un acteur malveillant. Le débat reste donc ouvert, la mise en place d'un tel système pour des réseaux de plusieurs milliers de noeuds étant bien sûr extrêmement intéressante d'un point de vue théorique mais demeurant tout aussi difficile à sécuriser correctement.

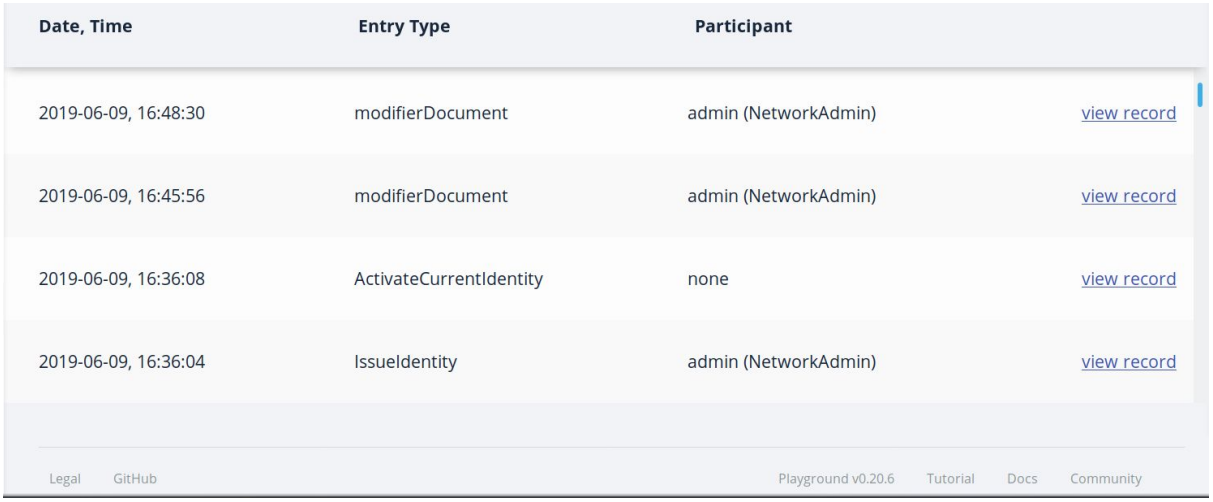
Fenêtre de création et visualisation des comptes "Utilisateur" par l'admin



Fenêtre de création et de visualisation des documents du système



Soumission d'une transaction de type modifierDocument



Historique des transactions inscrites dans la blockchain

Accès aux clients du système à partir du compte Philippe, pour tester les droits d'accès

Bibliographie

MOOC edX, Cours LFS171x : BLockchain for Business - An Introduction to Hyperledger Technologies :
https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS171x+3T2018/course/

Ressources de la fondation Ethereum France :
https://www.ethereum-france.com/

Ressources sur le projet Hyperledger :
https://www.hyperledger.org/

Informations sur la blockchain sur l'encyclopédie Wikipédia :
https://fr.wikipedia.org/wiki/Blockchain

Informations sur le Bitcoin :
https://bitcoin.fr/

Interview particulière de Vitalik Buterin, créateur de Ethereum : *https://www.ethereum-france.com/interview-de-vitalik-buterin-createur-dethereum-et-president-de-la-fondation-partie-1-sur-2/*

Ressource sur le système de vote mis en place par A. Ramachandran et M. Kantarcioglu avec Ethereum :
https://blockchain.ieee.org/images/files/images/clinicaltrialsforum-2018/U_of_Texas_WhitePaper.pdf

« Comprendre la Blockchain, anticiper le potentiel de disruption de la Blockchain sur les organisations », Livre blanc, Editeur U, janvier 2016.

Postérité

Il est possible de récupérer l'intégralité du code lié au projet sur le dépôt Git suivant: https://github.com/nathanael-denis/blockchain-vote

La branche la plus à jour étant la branche nommée "deploiement".

Dans le cadre du projet, nous avons suivi un MOOC sur la plateforme edX. Nous recommandons de suivre au moins la partie sur Hyperledger Composer pour comprendre comment déployer le réseau et le rôle des différents fichiers. Attention néanmoins, une fois inscrit, la période de disponibilité du MOOC est celle de la session. Il est fortement recommandé de garder une trace des procédures de déploiement du réseau, notamment des lignes de commande.
https://www.edx.org/course/blockchain-for-business-an-introduction-to-hyperledger-technologies

Quelques pistes de développement:

-Pour l'instant le projet fonctionne en local dans le navigateur; les votants sont interrogés un à un avec un système d'ouverture/fermeture de fenêtre. Il s'avère que Hyperledger Composer seul n'est pas suffisant pour pouvoir effectuer un déploiement plus réaliste - c'est-à-dire sur plusieurs machines-. Concrètement, si l'on souhaite utiliser un ordinateur par noeud, il est nécessaire de trouver un outil complémentaire. Une des pistes que nous avons trouvé vers la fin du projet est d'utiliser en complément un autre framework issu du projet Hyperledger: Hyperledger Fabric. La marche à suivre est détaillée dans la page suivante:

https://hyperledger.github.io/composer/latest/tutorials/deploy-to-fabric-single-org

-Il peut être intéressant de trouver un moyen de s'émanciper de l'interface graphique de Composer-Playground. Notamment, trouver un moyen de rendre l'insertion d'une chaîne de caractère transparente du point de vue de l'utilisateur, c'est-à-dire que la personne qui insère une chaîne ne devrait pas avoir à connaître les champs debutModification et finModification comme spécifié dans le fichier logic.js. Sur ce point, nous ne savons pas quelle démarche adopter.

-Trouver un moyen d'avertir les contributeurs qu'une modification doit être votée. Nous avions pensé à un mailing classique, mais cela est impossible avec le langage Javascript pour des raisons de sécurité. L'utilisation d'un timer pour limiter l'attente lors d'un vote serait alors possible. Dans notre version locale dans le navigateur, la méthode prompt() attend la réponse de l'utilisateur ce qui rend tout timer inutile.