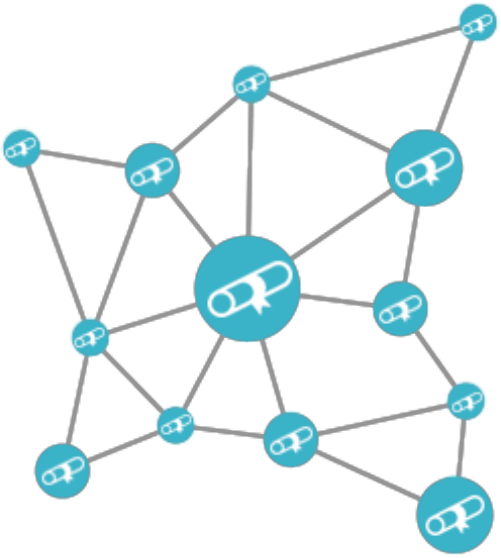
Projet UGAChain

Loris GENTILLON, Jordan JEAN, Enzo MOLION, Léo VALETTE



Enseignant :
Didier DONSEZ
Référent :
Anthony GEOURJON

Genèse du projet

D'après un rapport de Kroll Inc. , 30% des CV présenteraient des imprécisions et / ou seraient frauduleux (diplômes factices, événements non effectués).

Pour parer à cela, les entreprises dépensent du temps et de l'argent afin de déterminer la véracité des informations qui leurs sont données.

Cahier des charges

Ce projet a deux objectifs.
Le premier consiste en une refonte du projet UGAChain 2017 - 2018. Plus précisément, cet objectif consiste d'une part en la conception d'une application web mettant en oeuvre les dernières technologies avec l'aide du générateur JHipster pour remplacer l'application Java native conçu en 2018, et d'autre part la conception d'un réseau blockchain Hyperledger Fabric sous la version LTS 1.4.
Le deuxième objectif de ce projet consiste à fournir un générateur de code permettant d'obtenir un modèle de plateforme mettant en oeuvre une application web et une blockchain sur le modèle de UGAChain mais pour une utilisation plus générique. Ce deuxième objectif se présente ainsi en une contribution au projet JHipster.

Technologies employées

JHipster

Nous avons mis en place une application web tout à fait standard basée sur JHipster utilisant Angular et Bootstrap pour l'interface utilisateur, Spring pour le noyau fonctionnel, H2 en développement et PostgreSQL en production pour la base de donnée.

Hyperledger Fabric

Hyperledger est une plateforme open source de développement de blockchain. Ce projet a été initié en décembre 2015 par la fondation Linux. Le développement s'y fait essentiellement en langage Go. Elle regroupe différents *frameworks* permettant de développer des contrats intelligents ou des applications décentralisées dans la blockchain, à destination des entreprises.
Hyperledger est un projet "umbrella". Tous les projets Hyperledger sont conçus suivant une approche modulaire et extensible. Ils fournissent interopérabilité, des solutions sécurisés,

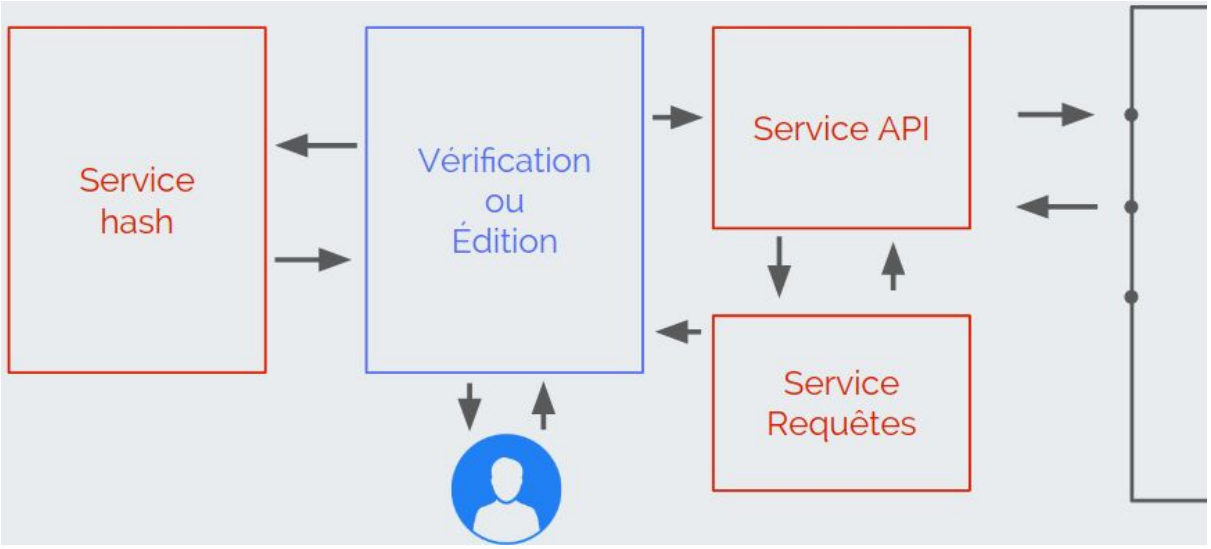
une approche token-agnostique (qui fonctionne peu importe le token utilisé), sans cryptomonnaie native, avec une facilité d'utilisation.[1]

Architecture et réalisation technique

Interface utilisateur

L'interface utilisateur (ou *frontend*) a été développé en Angular 7 à partir de celui généré par JHipster.
Ses responsabilités sont les suivantes :
- permettre à l'utilisateur de téléverser des fichiers correspondant aux diplômes à ajouter ou dont l'état doit être édité,
- afficher à l'utilisateur quelles requêtes sont en cours,
- afficher à l'utilisateur le résultat des requêtes.

Son architecture technique est la suivante :



L'utilisateur interagit avec les composants (*angular component*) de vérification ou d'éditions (ceux-ci étant paramétrés par le type d'édition correspondant - ajout, invalidation, revalidation, déclaration de fraude - puisque très similaires).
Au clic sur le bouton de requête par l'utilisateur, le composant sollicite un service (*angular service*) dédié à la gestion des hash qui se contente de calculer le hash SHA-256 de chacun des fichiers. Ensuite, il sollicite un service de gestion de l'API qui se charge de forger une requête et de l'envoyer au noyau fonctionnel via les points d'entrées API mais également à

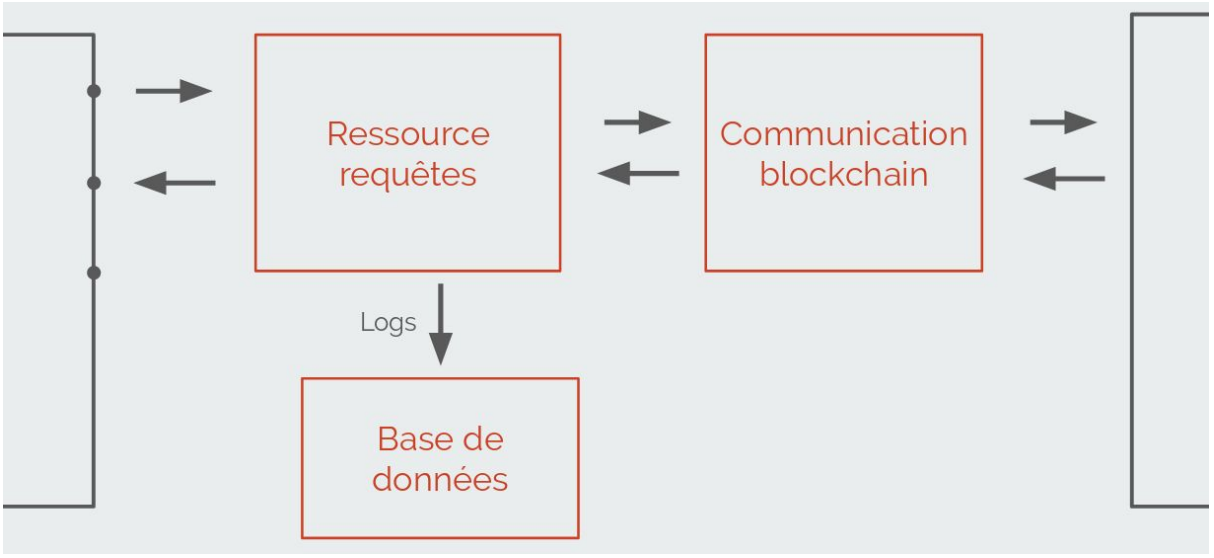
[1] https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf

un service de gestion des requêtes auquel sont abonnés les composants afin d'afficher les résultats à l'utilisateur.

Noyau fonctionnel

Les points d'entrées API

Les points d'entrées API, en partie été générés par JHipster, servent d'interface entre l'interface utilisateur et le module de communication blockchain. Lorsqu'une requête est reçue depuis l'interface utilisateur, si elle respecte le format, elle est transmise à la blockchain. Si la blockchain répond positivement à la requête, cette dernière est journalisée dans la base de données et le résultat est transmis à l'interface utilisateur. Sinon un message et un code d'erreur sont renvoyés et la requête n'est pas journalisée.
Le schéma suivant illustre ladite architecture :

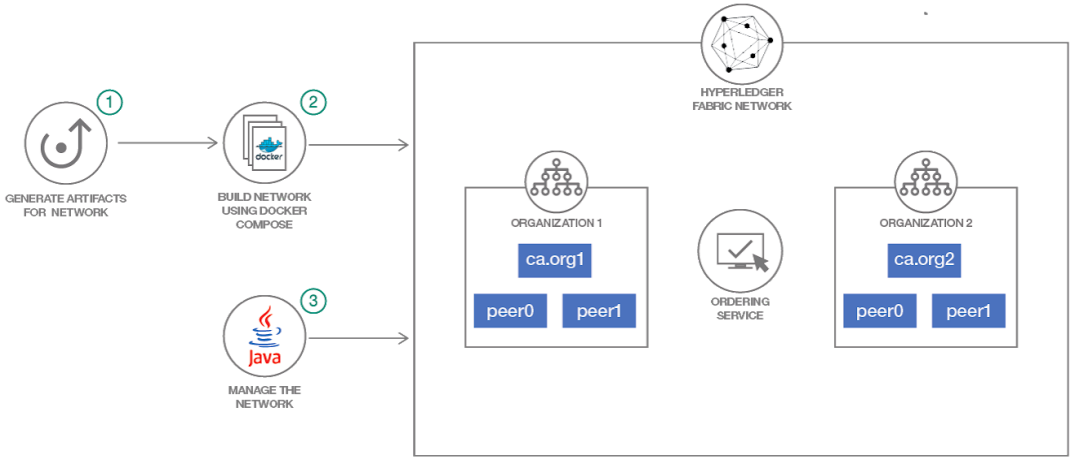


Le module de communication blockchain

Le module de communication blockchain est responsable de la communication entre le *back-end* de l'application web et le réseau Hyperledger. Celui-ci se présente sous la forme d'un *package* Java indépendant et est composé des *sub-packages* "request" et "blockchainException" et des objets Java permettant l'authentification et la connection au CA du réseau Hyperledger.
Le *package* "request" implémente les différentes requêtes qui peuvent être envoyées à la blockchain, ces dernières se présentent sous la forme des objets Java suivant : "Add", "Query", "SetValid", "SetInvalid" et "SetFraudulent". Ces derniers objets étendent tous l'objet "A_BlockchainRequest".
Le *package* "blockchainException" implémente les exceptions qui sont levées lors de situations particulières comme lorsqu'un hash donné n'a pas été trouvé dans la blockchain.

Blockchain

UGANetwork



Architecture et gestion du réseau Hyperledger
(source en bibliographie)

La blockchain mise en oeuvre pour UGAChain repose sur un réseau Hyperledger avec deux organisations ayant deux participants chacune. Le déploiement et l'utilisation de ce réseau se déroule de la façon suivante :

1. Génération des artéfacts par le script *byfn.sh* en utilisant les outils *cryptogen* et *configtx*.
2. Construction du réseau Hyperledger et des conteneurs Docker, puis installation et instanciation du smart contract *ugachaincode* sur la chaîne *diplomas*.
3. Utilisation de la blockchain par l'application web à travers le module de communication blockchain utilisant le SDK Fabric Java. On entend ici par utilisation de la blockchain l'envoi de requêtes du type ajout de hash, vérification de diplôme...

Le smart contract UGAChaincode

UGAChaincode est un *smart contract* de type *installed* ce qui signifie qu'il est exécuté avant que les transactions n'arrive au *ledger*. Ce dernier définit la logique métier mise en oeuvre au sein d'UGAChain. Plus précisément, c'est dans celui-ci que sont coder les fonctions permettant d'écrire et de lire les hashs des diplômes et leur état dans la blockchain en effectuant préalablement les vérifications nécessaires au respect de la logique métier.

Générateur Hyperledger pour JHipster

À la vue des possibilités d'application et des avantages offerts par les technologies de chaîne de blocs (par exemple crypto monnaie, transactions immobilières...), il est intéressant de proposer une solution pour facilement et économiquement générer un squelette d'application informatique mettant en oeuvre lesdites chaînes de blocs (à la manière de UGAChain) et permettant à de multiples clients de bénéficier des avantages de cette technologie sans avoir à investir toutes les ressources nécessaires à son développement, en particulier le temps.

Le générateur jhipster-blockchain que nous avons créé permet de répondre à ce besoin en ajoutant un réseau Hyperledger (sous la forme d'un répertoire "fabric-network") prêt à être déployé à la génération de JHipster, et en effectuant les changements nécessaires à l'application web pour que les entités résultantes de l'importation d'un JDL soient stockées aussi bien sur la base de donnée que sur Hyperledger.

Nous avons publié notre générateur sur le JHipster marketplace. Ainsi notre générateur peut être installé directement lors de la génération JHipster en sélectionnant le générateur jhipster-blockchain.

Gestion de projet

À la vue du cahier des charges ambitieux proposé par l'enseignant référent, nous avons planifié un premier *sprint*, volontairement long, visant à obtenir un prototype fonctionnel de bout-en-bout.
Celui-ci représentait une première preuve de la maîtrise des technologies complexes qui composent l'environnement technique de ce projet. Il nous a permis de pouvoir estimer le travail qui allait pouvoir être accompli dans le temps imparti et a duré 3 semaines.

Dans la continuité de ce premier *sprint* nous en avons planifié un deuxième visant à transformer notre prototype de bout-en-bout en un produit viable pouvant être mis en production. Etant donné qu'il ne nous restait que 2 semaines de projet, nous avons également passé l'objectif suivant en extension : le générateur d'application blockchain pour JHipster. Ce deuxième sprint a duré jusqu'à la fin du projet.
Cependant, étant donné que certaines tâches indépendantes ont été terminées plus tôt que prévu, il a été possible de détacher une personne pour réaliser l'extension évoquée précédemment avec l'aide d'une autre équipe projet.

Ainsi, nous étions parfaitement conscients qu'il était difficile de planifier à l'avance des tâches précises sur un projet contenant une telle diversité de tâches (synthèse du sujet, analyse du projet précédent, conception logicielle, développement). Nous avons donc structuré le projet autant que possible mais en restant agiles : phase planifiée de conception

réduite à une semaine, objectif d'obtenir un prototype fonctionnel de bout en bout le plus tôt possible, incrément sur celui-ci ensuite en *sprints* de granularité fine.

Outils collaboratifs

Nous avons utilisé les outils standard pour un projet de cette envergure : un projet git pour versionner notre code, un dépôt GitLab pour l'héberger, un tableau Trello pour l'organisation de notre tableau Kanban et Slack comme logiciel de messagerie.

Métrique logicielle

Nous commençons par donner les métriques qui nous semblent les plus pertinentes.
Ce projet représente un effort de 626 heures au total, soit 4 mois à raison de 35h/semaines.

Les langages de codes utilisés ont la répartition suivante : 52.24% de Java, 33.17% de TypeScript, 9.62% de HTML, 2.57% de JavaScript et 2.32% de CSS.

Ayant utilisé des générateurs de code, il est peu pertinent d'exprimer ce projet en nombre de lignes de codes. Cependant, nous avons procédé à un compte en expurgeant au maximum les lignes de codes générées et nous arrivons au compte suivant : 4 dépôts GitLab, 148 fichiers modifiés, 14094 insertions, 8164 suppressions.

Bibliographie

https://fr.wikipedia.org/wiki/Hyperledger
https://github.com/IBM/blockchain-application-using-fabric-java-sdk
https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf
https://marmelab.com/blog/2016/05/12/blockchain-expliquee-aux-developpeurs-web-la-theorie.html
https://blockchainfrance.net/2018/05/22/comprendre-les-tokens/
https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf
https://github.com/RICM5-BlockChain
https://air.imag.fr/index.php/RICM5_2017_2018_-_UGAChain_/_SRS#Horizon
https://crypto.stackexchange.com/questions/47809/why-havent-any-sha-256-collisions-been-found-yet
https://apereo.github.io/cas/6.0.x/protocol/CAS-Protocol.html
https://github.com/jhipster/jhipster-sample-app-oauth2
https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth
https://openblockchain.readthedocs.io/en/latest/API/CoreAPI/#rest-api
https://github.com/hyperledger/fabric-sdk-java
https://hyperledger-fabric.readthedocs.io/en/release-1.4/fabric-sdks.html
https://hyperledger-fabric.readthedocs.io/en/release-1.4/developapps/application.html
https://humanwhocodes.com/blog/2012/05/08/working-with-files-in-javascript-part-1/
https://humanwhocodes.com/blog/2012/05/15/working-with-files-in-javascript-part-2/
https://github.com/IBM/blockchain-application-using-fabric-java-sdk
https://hyperledger-fabric.readthedocs.io/en/release-1.4/network/network.html
https://medium.com/@lkolisko/hyperledger-fabric-sdk-java-basics-tutorial-a67b2b898410
https://docs.spring.io/spring-security/site/docs/current/reference/html/jc.html
https://codeburst.io/a-concise-tutorial-on-working-with-hyperledger-fabric-java-sdk-a6f11d8bb5b0
https://angular.io/tutorial/
https://www.jhipster.tech/jdl/
http://reactivex.io/rxjs/manual/overview.html
https://perhonen.fr/blog/2015/05/un-reverse-proxy-apache-avec-mod_proxy-1713
https://blog.victor-hery.com/2012/10/configurer-reverse-proxy-apache.html#edit-comme-promis-la-version-https